FINAL REPORT

ON A

CODING SYSTEM DESIGN FOR ADVANCED SOLAR MISSIONS

Contract NAS2-3637

Submitted to:

AMES RESEARCH CENTER

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

December 18, 1967

CODEX

CODEX CORPORATION • 222 ARSENAL STREET, WATERTOWN, MASSACHUSETTS 02172

FINAL REPORT

on a

CODING SYSTEM DESIGN FOR ADVANCED SOLAR MISSIONS

Contract NAS2-3637

Submitted to:

Ames Research Center
National Aeronautics and Space Administration

December 20, 1967

Submitted by:

Codex Corporation
222 Arsenal Street
Watertown, Massachusetts 02172

Table of Contents

Table of Contents (con.) Page

Introduction

This is the Final Report on Contract NAS2-3637, "Coding System Design for Advanced Solar Missions." Work reported in detail in earlier reports is summarized in the following section; that done on this phase of the contract is then summarized in the next section, with detailed results appearing in three appendices.

Summary of Earlier Reports

Shortly after the beginning of this contract, Ames carried out in-house simulations which showed that efficiencies predicted theoretically for sequential decoding schemes [Phase II Report, Appendix A, 1966] could be largely obtained with a relatively simply implemented code [Interim Report, 1966]. As simulations of the best of the competitive OED schemes showed significantly inferior (2-3 db) performance, with no implementation advantage, Ames quickly focused all effort on sequential decoding and pressed toward its implementation on future Pioneer spacecraft. In previous reports [Interim Report, 1966, and Second Interim Report, 1967] we described our contributions to this effort, which mainly consisted of detailed support in certain areas for the central development at Ames. Besides the OED simulators, we wrote efficient machine language decoding programs for the SDS 910-920 and the IBM 7094; the principles were described in the Interim Report, while the SDS program was flow-charted and listed in the Second Interim Report. A convolutional encoder intended to be suitable for incorporation into the spacecraft was designed and breadboarded. In the Second Interim Report, we suggested possible approaches to the problems of input/output and timing synchronization, and error detection. We wrote basic machine language programs for a number of small general-purpose computers, in order to compare the speeds which could be obtained. Finally, we did a detailed paper design of a special-purpose sequential decoder for the Pioneer application, to determine the maximum easily attainable computation speed (1 μs) and cost ($10,000 for materials).

Summary of Recent Studies

In the final phase of Contract NAS2-3637, which is the subject of this report, we shifted our vision to the more distant future. We assume that minimizing the power required to support a given bit rate will continue to be the principal goal of the communications system designer; we also assume that the error probability required will become lower and lower as more and more data reduction is performed on board. In the present study, we also have assumed the feasibility of coherent demodulation, which today would imply data rates of more than 10-100 bps, and have therefore been able to assume the applicability of the ideal white gaussian noise channel model.

Under these conditions, sequential decoding schemes seem to be capable of approaching the theoretically determined optimal efficiency to within about a factor of 2, with an arbitrarily low undetected error probability. Practically speaking, such schemes can be made about 3-4 db from optimum. Combined with this performance are systems advantages, such as relatively simple on-board equipment, simple (PSK) modulation and demodulation, moderate decoding requirements (much of which can be off-line), and occurrence of errors in distinct clusters (frames), effectively all of which can be detected. It therefore seems likely that sequential decoding schemes will be the standard for some time to come.

However, that 3 db of inefficiency remains, and the question arises of whether it is possible to improve on the performance of sequential decoding by going to some more complicated scheme. Such a scheme, we feel, will very likely be some elaboration of sequential decoding, in which the additional complication serves to modify the sequential decoder's computational behavior, which is its fundamental limitation. A particular example of such a scheme is already known — called by us 'concatenation with sequential decoding', and by Falconer [1967], who studied the scheme in his doctoral thesis, 'hybrid sequential and algebraic'. It is easy to show that in principle concatenation can allow as close to optimum efficiency as desired, and in fact to exhibit a scheme which approaches within about 1 db of optimum without outrageous complexity [Phase II Report, Appendix E, 1966]. In the presence of an urgent require-

ment, one could proceed to implement such a scheme today; all the major elements are well understood, and the principal task would be to settle on specific strategies and parameters and evaluate complexity and performance in detail.

In the work reported here, we have taken a more leisurely approach, in an attempt to understand better what is going on, and perhaps to develop a simpler scheme that the above, which seems in many respects brute force and inelegant. Our work consisted of three principal parts, reported in Appendices A, B and C. First, we have reviewed the fundamental causes of the decoding behavior of a general class of decoders for convolutional codes, which includes sequential decoders as a typical subclass, and have developed new understanding about the cause of the computational distribution, the character of significant noise bursts and the resulting error patterns, and the effects of finite constraint length. Second, we have used these results in the development of a point of view about concatenation schemes, and have used this point of view to develop a number of classes of different schemes and to predict the most important aspects of their computational behavior. Third, we have succeeded in substantially validating this point of view by simulating a few simple concatenation schemes and comparing the observed performance with predictions.

Appendix A is actually a rather complete review of what is known about the performance of random tree codes (a convolutional-like code suitable for analysis) with optimum decoders, a class into which sequential decoders apparently fall. It contains:

1. Upper and lower bounds on the performance of tree codes, and asymptotically exact expressions for the performance of random tree codes at all rates.

2. A demonstration that, when converted into block codes, random tree codes equal the performance of random block codes and are therefore optimum block codes at high rates.

3. A demonstration that an optimum (maximum likelihood) decoder for a tree code can be significantly simpler than that required for a block code which has the same performance, and thus that in a fundamental

sense tree codes are better than block codes.

4. A discussion of the character of error patterns with finite constraint length codes, and of the noise bursts which cause them, showing that for large constraint lengths a single type of pattern becomes dominant.

5. An optimum, variable-computation decoding algorithm for large constraint length codes which exhibits a Pareto computational distribution and illustrates the causes thereof.

6. A consideration of the close connection between the event of decoding error with finite constraint length codes and the event of computational overflow.

7. An heuristic estimate of the computational distribution to be expected with sequential decoding of finite constraint length codes.

8. Suggestions for sequential decoder operation, including moderate constraint lengths and automatic resynchronization.

Appendix B depends primarily upon the observation in 4, above, and shows how it may be used to predict the most important parameter of concatenation-type schemes, the overall Pareto exponent. Appendix B contains:

1. Justification for considering only a single type of noise pattern in analyzing computational probabilities.

2. A number of different concatenation-type schemes, with analysis of the Pareto exponent and rate loss of each. In every case the rate loss can be made to approach zero while the Pareto exponent of the concatenated scheme remains at some fixed multiple of the original Pareto exponent.

Finally, some simulations intended to verify the analysis techniques of Appendix B and yield quantitative indications of the results to be expected are reported in Appendix C. It contains:

1. Descriptions of the basic sequential decoding schemes simulated, and of two simple concatenation-type schemes.

2. Computational distributions and error probabilities obtained with these schemes in million-bit runs at varying Pareto exponents and constraint lengths.

3. An assertion that the analysis techniques of Appendix B and the finite constraint length predictions of Appendix A seem substantially valid, and consideration of some discrepancies.

4. A quantitative estimate of the Pareto distribution coefficient, and, assuming its validity, a comparison of the decoder parameters required to achieve certain efficiencies with and without concatenation.

In conclusion, what we now have is a good basic understanding of sequential decoders, and the ability to predict fairly accurately the most important features of the performance and complexity of proposed schemes. What we do not have is an elegant, economical scheme with a large exponent multiplier (performance improvement vis-a-vis ordinary sequential decoding); at the moment we must be content with elaboration of the basic concatenation scheme described above. With the exception of backwards-forwards schemes with their moderate improvement, we also do not have the quantitative results about any particular scheme which would be necessary to proceed to implementation. Our recommendations would be to proceed on these two fronts, with the distribution of effort between them depending on the urgency of actually developing concatenation schemes for practical application.

References

Codex Corp., Phase II Report on a Study of Coding for Deep Space Telemetry, Contract NAS2-2874, Watertown, Mass., March 28, 1966.

Codex Corp., Interim Report on a Coding System Design for Advanced Solar Missions, Contract NAS2-3637, Watertown, Mass., October 20, 1966.

Codex Corp., Second Interim Report on a Coding System Design for Advanced Solar Missions, Contract NAS2-3637, Watertown, Mass., May 12, 1967.

D. D. Falconer, "A Hybrid Sequential and Algebraic Decoding Scheme", Ph.D. Thesis, M.I.T. Dept. of Electrical Engineering, February, 1967.

APPENDIX A

REVIEW OF RANDOM TREE CODES

Introduction

The use of convolutional codes with sequential decoding is becoming the standard method of obtaining the most efficient communication on memoryless channels. Extensive analyses preceded this use, in a rather rare example of communication theory inspiring a practical application. The analyses have principally dealt with the Fano algorithm for sequential decoding, and with a type of tree code called a random tree code, to be introduced below; simulations have confirmed that the results of these analyses are qualitatively accurate when applied to convolutional codes, which are the tree codes of practical interest.

Two important recent papers by Viterbi (1967) and Jacobs and Berlekamp (1967) have suggested that the predicted and observed performance of sequential decoding is principally due to the structure of the random tree codes themselves, and would be observed with any of a broad class of decoding algorithms. We are therefore motivated to study the properties of random tree codes which depend on a minimum of assumptions about the decoding process; this study shall be the subject of this appendix.

The most important results reported here are contained in the two papers referenced above. What we have done is to extend these earlier results in some directions and try to draw out some of their intriguing consequences. We examine minutely the structure of random tree codes and their performance with maximum likelihood decoding, obtaining exact results for the error probability at all rates. These results agree with the results obtained for sequential decoding and with a bound on the best possible performance at high rates. We exhibit a maximum likelihood decoding algorithm which has in many respects the same type of decoding complexity as sequential decoding, namely the Viterbi algorithm, and by various modifications show that the correspondence may be made

almost exact, at least at high rates and in the asymptotic limit. From the results obtained, we make several suggestions about how to use sequential decoding which are at variance with conventional practice. We also hope by our analyses to have provided additional insight into how and why sequential decoding works as it does, so that we can judge the effects of possible modifications, and into what factors are responsible for its limitations, so that we can perhaps surmount them.

Though providing insight into sequential decoding is our principal goal, we develop some interesting subsidiary results. We show exactly how tree codes are better than block codes by showing that, when made into block codes, tree codes give just as good performance, but admit less complex decoding algorithms, while when used naturally as tree codes, they give better performance at the same rates. We show that this superiority depends on suspending final decoding decisions for much longer than a constraint length and is therefore not obtained with decoders which make decisions on the basis of a single constraint length; we estimate what the effective decoding constraint length must be if optimum performance is not to be degraded. We show that most decoding error runs cluster around a typical length which depends on rate, the clustering becoming more pronounced with increasing constraint length. Finally, we give a modification of the Viterbi algorithm suitable for a tree code which is not resynchronized with a known constraint length of information symbols.

Before getting into the body of the paper, we establish some definitions and review pertinent block code results. We then introduce tree codes, show how they may be made into block codes, and discuss the relationships between their performances which necessarily result. We spend some time with the concept of merging, which is central to the understanding of tree codes, and introduce a variant of the tree picture, called a trellis. We proceed to define and analyze random tree codes, determining their error probability with maximum likelihood decoding; we then discuss the character of typical error patterns. The Viterbi algorithm is introduced and shown to be equivalent to maximum likelihood decoding and therefore optimum. Variants of the Viterbi algorithm involving changes in the decoding constraint length are presented, leading

to a scheme very similar in behavior to sequential decoding. We review the results of Jacobs and Berlekamp to establish the close connection between the event of buffer overflow with an infinite constraint length code with the event of decoding error with a finite constraint length code. We consequently suggest that it may be profitable to match the code constraint length to sequential decoder buffer size in order to minimize output error probability; we estimate the computational distribution of a finite constraint length sequential decoder, and suggest that buffer overflows may be made rare without gross effect on the over-all error probability. Finally, we show that there are no fundamental problems in using an unterminated tree code with the Viterbi algorithm, and suggest that the same may be true with sequential decoders.

Asymptotic Equivalence

We shall mostly be interested in expressions which are valid when the constraint length or some other quantity becomes very large. It will be convenient to introduce notation for equalities and inequalities valid in this asymptotic range. Let A be the quantity that is becoming large, and let P be the quantity we are interested in. P is said to be asymptotically equal to A^{-E} if for any $\epsilon > 0$ there is some value of A sufficiently large that

$$A^{-(E+\epsilon)} < P < A^{-(E-\epsilon)}, \tag{1}$$

E is called the exponent of P, where the quantity A to which this exponent has reference will be clear from the context. We introduce the notation:

$$P \doteq A^{-E}. \tag{2}$$

Similarly, we say that P is asymptotically less than or greater than A^{-E} if one or the other of the inequalities in (1) holds, and introduce the notation

$$P \dot{\leq} A^{-E}; \qquad P \dot{\geq} A^{-E}; \tag{3}$$

then, for example, we can write briefly

$$P \doteq A^{-E} \quad \text{iff} \quad P \dot{\leq} A^{-E} \text{ and } P \dot{\geq} A^{-E}. \tag{4}$$

Some Useful Quantities

We now introduce Gallager's (1965) $E_o(\rho)$ function, which appears repeatedly in the analysis of codes on the discrete memoryless channel. Let the channel be defined by its input alphabet x_k, its output alphabet y_j, and the transition probability matrix p_{jk} giving the probabilities that if x_k is sent, y_j will be received. Define also an input distribution p_k, to be thought of as the probabilities of the input letters x_k. Then Gallager's function is defined as

$$E_o(\rho) = -\ln \sum_j \left[\sum_k p_k p_{jk}^{\frac{1}{1+\rho}} \right]^{1+\rho}, \qquad \rho \geq 0. \tag{5}$$

We shall find that many results can be conveniently expressed in terms of the family of functions $T_\rho(R)$, defined by

$$T_\rho(R) = E_o(\rho) - \rho R, \tag{6}$$

where R is to be thought of as a code rate. As a function of R, $T_\rho(R)$ is a straight line of slope $-\rho$, which equals $E_o(\rho)$ when R=0 and which equals 0 at $R=R_\rho$, where R_ρ is defined by

$$R_\rho = E_o(\rho)/\rho . \tag{7}$$

In terms of R_ρ we can rewrite (6) as

$$T_\rho(R) = \rho(R_\rho - R) \tag{8}$$

Gallager (1965) has shown that $E_o(\rho)$ increases and R_ρ decreases monotonically with ρ.

Figure 1 illustrates a typical set of $T_\rho(R)$ for $\rho = \epsilon$, 1/2, 1, 2, and $1/\epsilon$. The channel used for illustration is

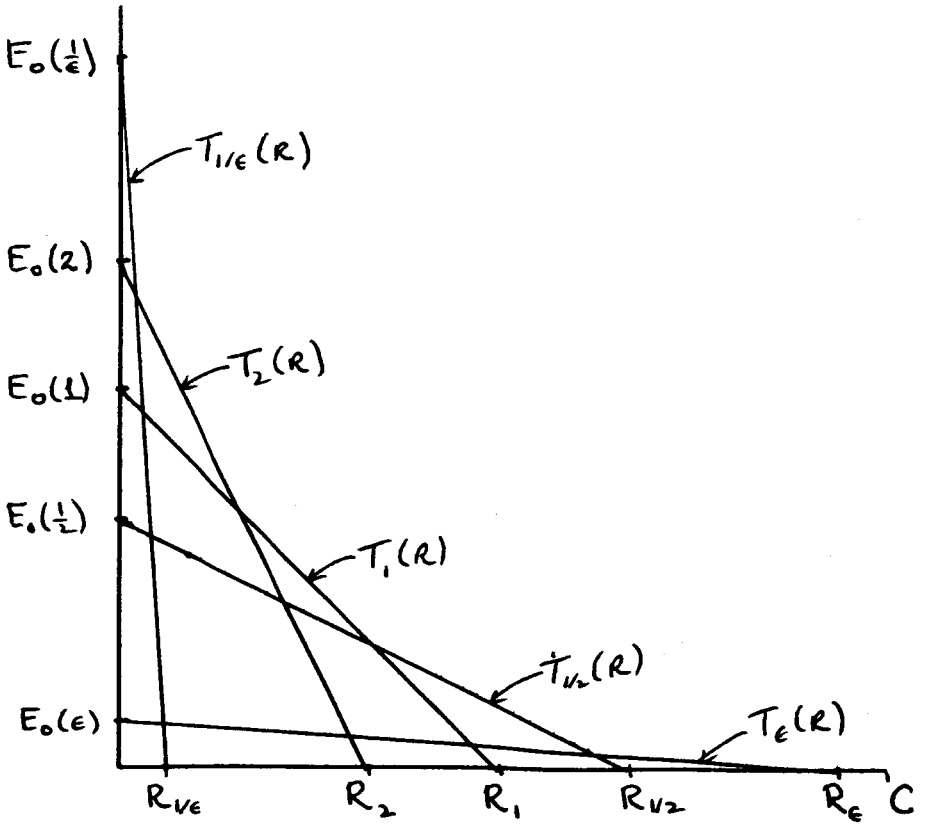


Figure 1. Typical $T_\rho(R)$ (Very Noisy Channel)

the very noisy channel of capacity C, for which

$$E_o(\rho) = \frac{\rho}{1+\rho} C. \tag{9}$$

Finally, the sphere-packing exponent $E_{sp}(R)$ is defined as the upper envelope of all the functions $T_\rho(R)$:

$$E_{sp}(R) = \max_{\rho \geq 0} T_\rho(R). \tag{10}$$

Gallager has shown that the sphere-packing exponent can be written parametrically in terms of ρ as

$$\begin{aligned} E_{sp}(\rho) &= E_o(\rho) - \rho E_o'(\rho), \\ R(\rho) &= E_o'(\rho), \end{aligned} \tag{11}$$

where $E_o'(\rho)$ is the derivative of $E_o(\rho)$ with respect to ρ.

Some of these quantities appear sufficiently frequently to be given special names. The most important are the channel capacity C, the computational cutoff rate R_{comp}, and the critical rate R_{crit}, defined by

$$\begin{aligned} C &= E_o'(0) = R(0) = R_o; \\ R_{comp} &= R_1; \\ R_{crit} &= E_o'(1) = R(1). \end{aligned} \tag{12}$$

Block Code Results

A block code of length N and rate R (in units of nats) for a discrete memoryless channel is defined as any collection of M = exp NR sequences of N input letters, called the code words, x_m.

A received word y is any sequence of N output letters. A decoding algorithm is any procedure for assigning a code word x_m to each received

sequence; a decoding error occurs whenever the code word so chosen is not the one actually transmitted. Shannon, Gallager, and Berlekamp have shown (1967) that for N sufficiently large all block codes of rate R have probability of error Pr($\mathcal{E}$) bounded by

$$\Pr(\mathcal{E}) \doteq \exp -NE_{opt}(R), \tag{13}$$

where

$$\begin{aligned} E_{opt}(R) &= E_{sp}(R), \quad R \geq R(\rho_x); \\ &= T_{\rho_x}(R), \quad R \leq R(\rho_x), \end{aligned} \tag{14}$$

where $\rho_x \geq 1$ is defined by

$$E_o(\rho_x) = -\sum_k \sum_{k'} p_k p_{k'} \ln \left[\sum_j p_{jk}^{1/2} p_{jk'}^{1/2} \right], \tag{15}$$

and $E_o(\rho)$, $T_\rho(R)$, $E_{sp}(R)$, and $R(\rho)$ are defined in the previous section by (5), (6), (10), and (11). A typical $E_{opt}(R)$ curve appears in Figure 2.

A list-of-L decoding algorithm for a block code is one in which L code words x_m are assigned by the decoding algorithm to each received word y; a list decoding error occurs whenever the transmitted word is not on the list. Shannon, Gallager, and Berlekamp (1967) also show that if

$$L \doteq \exp NR_L, \tag{16}$$

where R_L is the list rate, then

$$\Pr(\mathcal{E}) \doteq \exp - NE_{sp}(R - R_L). \tag{17}$$

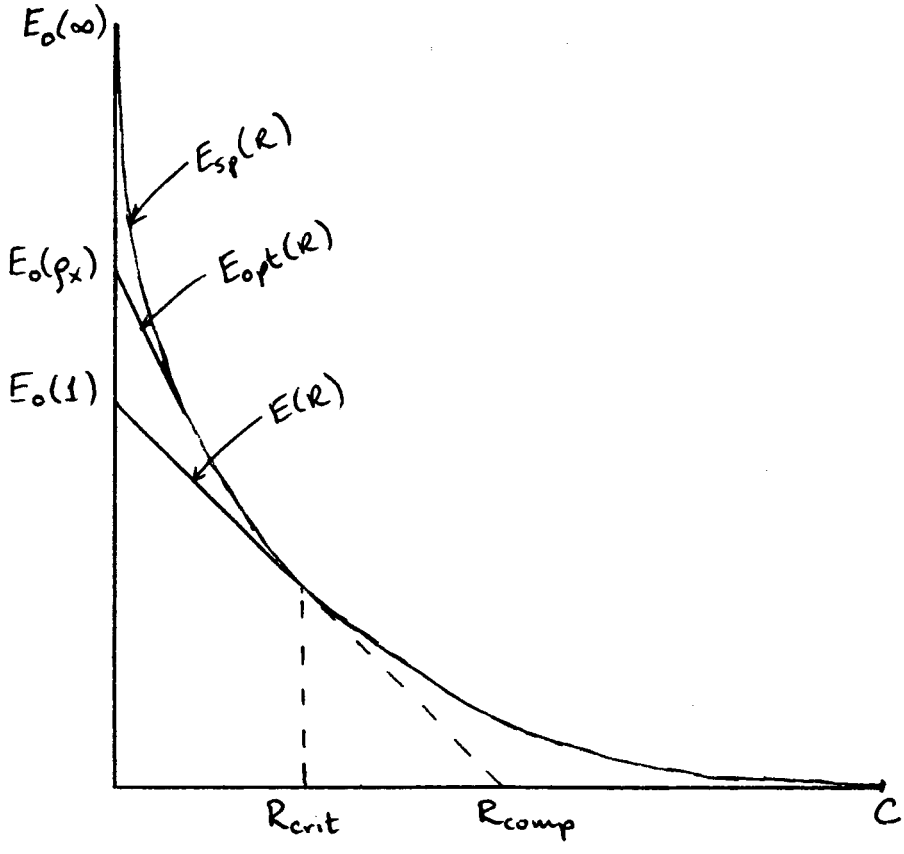


Figure 2. Important Exponents

$E_{sp}(R)$: Sphere-Packing Exponent

$E_{opt}(R)$: Best Possible Block Code Error Exponent

$E(R)$: Error Exponent for Random Block Codes.

Comparison of (17) with (13) and (14) shows that (17) is somewhat weaker when $L = 1$ and $R \leq R(\rho_x)$. The sphere-packing exponent which appears here is illustrated in Figure 2.

The above bounds give limits on how well one can do with any code and decoding algorithm. Random block codes with maximum likelihood decoding meet these limits at the higher code rates. A random block code is one in which the input letters of the various code words are picked from a random ensemble in such a way that:

1. The probability that x_k appears in any position of any code word is p_k, where the p_k represent the input distribution.
2. The probability that x_k appears in any position of any code word, given the input letters which appear in that position in any combination of other code words, is p_k.
3. The probability that x_k appears in any position of any code word, given the input letters which appear in any combination of other positions in any combination of other code words, is p_k.

These conditions can be met, for example, by choosing each of the N letters of each of the M code words independently and at random from an ensemble in which the probability of picking x_k is p_k.

Next, a decoding algorithm is called maximum likelihood if it assigns to a received word y that code word x_m for which the probability of y given x_m, $\Pr(y \mid x_m)$, is maximum; a maximum likelihood decoder minimizes the probability of error if the code words are equally likely, as can generally be assumed.

For random block codes with maximum likelihood decoding, Gallager (1965) proved that the probability of error is asymptotically equal to

$$P_r(\varepsilon) \doteq \exp -NE(R), \tag{18}$$

where

$$E(R) = E_{sp}(R), \quad R \geq R(1) = R_{crit};$$
$$= T_1(R) = R_{comp} - R, \quad R \leq R(1). \tag{19}$$

[Actually, Gallager only proved $\leq$, but the converse is easily shown by the methods of Shannon, Gallager and Berlekamp (1967).] A typical E(R) curve is illustrated in Figure 2. It is clear that above $R(1) = R_{crit}$,

$$E(R) = E_{opt}(R), \quad R \geq R_{crit}, \tag{20}$$

so that in this range the random block code is asymptotically optimum.

Gallager's proof actually requires somewhat weaker conditions on the random code than were stated above. In particular, Condition 2 may be replaced by the following Condition 2' (pairwise independence):

2'. The probability that x_k appears in any position of any code word, given the input letter which appears in that position in any other code word, is p_k.

That this weaker condition is permissible will be important in the analysis of random tree codes, as was pointed out by Viterbi.

Decoding Complexity

With maximum likelihood decoding, the decoding complexity G is proportional to the total number of code words M = exp NR, if the likelihood for each code word must be computed. Certainly asymptotically

$$G \doteq \exp NR. \tag{21}$$

We can therefore express the random block code error probability, for example, in terms of the decoding complexity G as

$$P_r(\mathcal{E}) \doteq G^{-\frac{E(R)}{R}}, \tag{22}$$

by substitution of (21) in (18). Thus while the error probability decreases exponentially with block length, it decreases only algebraically with the decoding complexity, with complexity exponent E(R)/R.

Tree Codes

We now arrive at the subject of this review, tree codes. In this section we shall define tree codes, introduce terminated tree codes, and show that the latter may be considered as block codes.

With block codes we made no assumptions about the data source, save that there were M messages to be encoded. With tree codes it is necessary to assume that the data source supplies an ordered sequence of source letters $s_1, s_2, s_3 \ldots$, where each letter comes from an alphabet of size q. Much data appears naturally as a serial stream of binary bits; these can be considered as individual letters, in which case q=2, or can be taken t at a time, say, in which case $q=2^t$. A tree encoder codes into an ordered sequence of branches x_i, where each branch consists of b channel input letters, and one branch is put out for each source letter in. The tree code rate r in nats per source letter is then equal to

$$r = \ln q/b. \tag{23}$$

A tree code is further characterized by its encoding constraint length ν, which has the significance that the branch x_i is a function only of the $\nu + 1$ source letters $s_{i-\nu}, \ldots, s_i$. (Note that here the units of ν are source letters, a convention which differs from that in some of the convolutional coding literature; also, the constraint length is for notational convenience one less than usual.) One observes that by taking $\nu = 0$, b = N, q = exp NR, and thus r = R, we obtain a block code of length N and rate R, so in this rather uninteresting sense block codes are included in the class of tree codes; thus tree codes must be as good as block codes in general.

A much more interesting way of constructing a block code from a tree code is to terminate the tree code as follows. Let the encoder insert after every K source letters a resynchronizing sequence of ν fixed,

dummy source letters, also known to the decoder. The first branch x_1 after the resynchronizing sequence is then a function only of the corresponding source letter s_1 and the dummy resynchronizing sequence, not of any previous source letters; similarly, no succeeding branch depends on source letters before s_1. Consequently, the $K+\nu$ branches $x_1,\ldots, x_{K+\nu}$ depend only on the K source letters $s_1,\ldots s_K$, so that successive sequences of $K+\nu$ branches form independent blocks. In this way, a terminated tree code is made into a block code. The block code has length $N = b(K+\nu)$ channel inputs, $M = q^K$ code words, and therefore block code rate

$$\begin{aligned} R &= \frac{\ln M}{N} \\ &= \frac{Kr}{K+\nu} \\ &= \lambda r, \end{aligned} \tag{24}$$

where we have defined λ, the synchronization rate loss, by

$$\lambda = \frac{K}{K+\nu}. \tag{25}$$

In what follows, we are going to find expressions for the probability of error of terminated tree codes which will be exponentially decreasing with the constraint length ν and will thus take the general form

$$\Pr(\mathcal{E}) \doteq \exp - \nu b e(r). \tag{26}$$

where e(r) will be the tree code exponent. Since these are also block codes, we can also write (26) in terms of the block code parameters as

$$\Pr(\mathcal{E}) \doteq \exp - NE(R), \tag{27}$$

where N is the block code length, R the block code rate, and the error component E(R), from comparison of (26) and (27), is equal to

$$\begin{aligned} E(R) &= \frac{\nu_b e(r)}{N} \\ &= (1-\lambda)\, e(r) \\ &= \left(1-\frac{R}{r}\right) e(r), \end{aligned} \tag{28}$$

where we have used (24) and (25).

In Figure 3, we give the geometrical construction of E(R) from e(r) implied by (28). We see that by proper choice of the synchronization rate loss λ in the range $0 \leq \lambda \leq 1$ we can obtain from a tree code with tree code exponent e(r) a block code with any block code rate and exponent pair on the straight line given by (28).

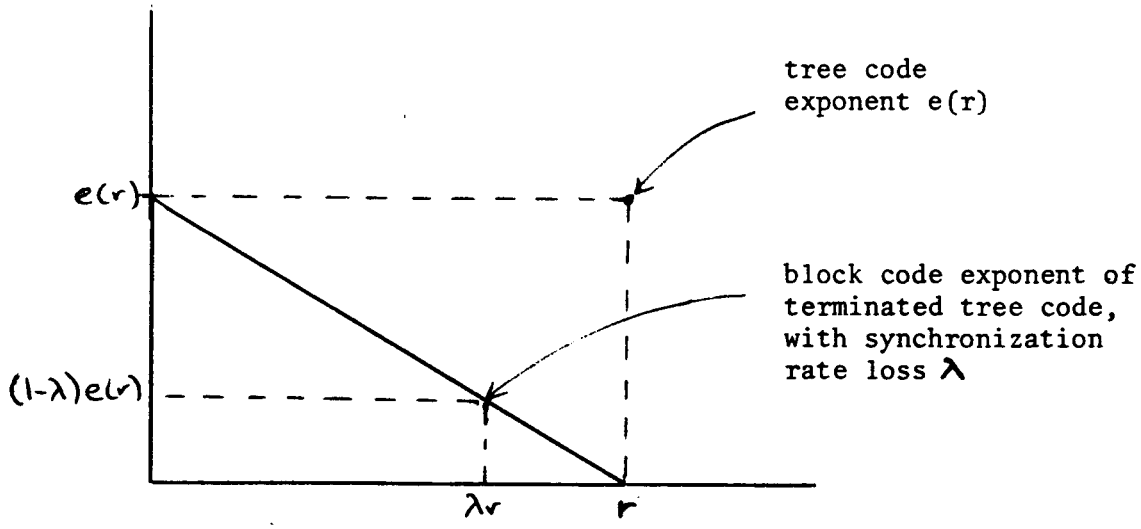


Figure 3. Block Code Exponent vs. Rate Achievable with a Terminated Tree Code of Tree Code Rate r.

This relationship between tree code and block code exponents is very important. As an example of its power, we now derive a bound on the tree code exponent, which depends on the fact that the terminated tree code considered as a block code cannot have a block code error exponent better than $E_{opt}(R)$, the error exponent of the best possible block code. We recall from (14), (10), and (6) that

$$\begin{aligned} E_{opt}(R) &= \max_{0 \le \rho \le \rho_x} T_\rho(R) \\ &= \max_{0 \le \rho \le \rho_x} [E_o(\rho) - \rho R], \end{aligned} \tag{29}$$

where ρ_x was defined in (15). Let us then determine the maximum possible tree code exponent for some tree code rate r. First, let $\rho(r)$ be the ρ such that $R_\rho = r$, and suppose $r \ge R_{\rho_x}$, so that $\rho(r) \le \rho_x$. Recall that the rate R for which the maximum in (29) occurs at $\rho = \rho(r)$ is given by $R[\rho(r)] = E_o'[\rho(r)]$. Define the optimum tree code exponent $e_{opt}(r)$ by

$$e_{opt}(r) = E_o[\rho(r)], \quad R_{\rho_x} \le r \le C. \tag{30}$$

We now show that the actual tree code exponent e(r) must be bounded by $e_{opt}(r)$. For suppose that

$$\begin{aligned} &e(r) > E_o[\rho(r)]; \\ &r = R_{\rho(r)}, \end{aligned} \tag{31}$$

then a terminated tree code of block code rate $R[\rho(r)]$ would have, from (28), the equivalent block code exponent

$$
\begin{aligned}
E\{R[\rho(r)]\} &= \left\{1 - \frac{R[\rho(r)]}{r}\right\} e(r) \\
&> \left\{1 - \frac{R[\rho(r)]}{r}\right\} E_o[\rho(r)] \\
&= E_o[\rho(r)] - \rho(r) R[\rho(r)] \qquad (32) \\
&= E_{opt}\{R[\rho(r)]\}
\end{aligned}
$$

where we have substituted (31) and (7) and used the definition of $R[\rho(r)]$. But to have $E\{R[\rho(r)]\} > E_{opt}\{R[\rho(r)]\}$ is impossible. Thus

$$e(r) \leq e_{opt}(r), \; R_{\rho x} \leq r \leq C. \qquad (33)$$

At low rates, it is easy to see by the same reasoning that the maximum tree code exponent is the constant

$$e_{opt}(r) = e_{opt}(R_{\rho x}) = E_o(\rho_x), \; 0 \leq r \leq R_{\rho x}, \qquad (34)$$

for otherwise we would be able to obtain a zero-rate block code with exponent better than $E_o(\rho_x) = E_{opt}(0)$. Thus with the parametric expression implied by (31),

$$
\begin{aligned}
e_{opt}(\rho) &= E_o(\rho) \\
r(\rho) &= R_\rho ,
\end{aligned}
\qquad (35)
$$

we have completed a bound on e(r) for all rates. This bound was obtained by Viterbi (1967) through a different argument.

Undoubtedly the easiest way to understand this argument is to examine the associated geometrical construction of $e_{opt}(r)$ from $E_{opt}(R)$. This construction, which is the inverse of that of Figure 3, appears in Figure 4. From each of the tangents to the $E_{opt}(R)$ curve, we obtain the point on the $e_{opt}(r)$ curve which completes the rectangle. It should be obvious that any tree code exponent lying in the region above the $e_{opt}(r)$ curve would give by the construction of Figure 3 a block code exponent which would somewhere lie above the $E_{opt}(R)$ curve.

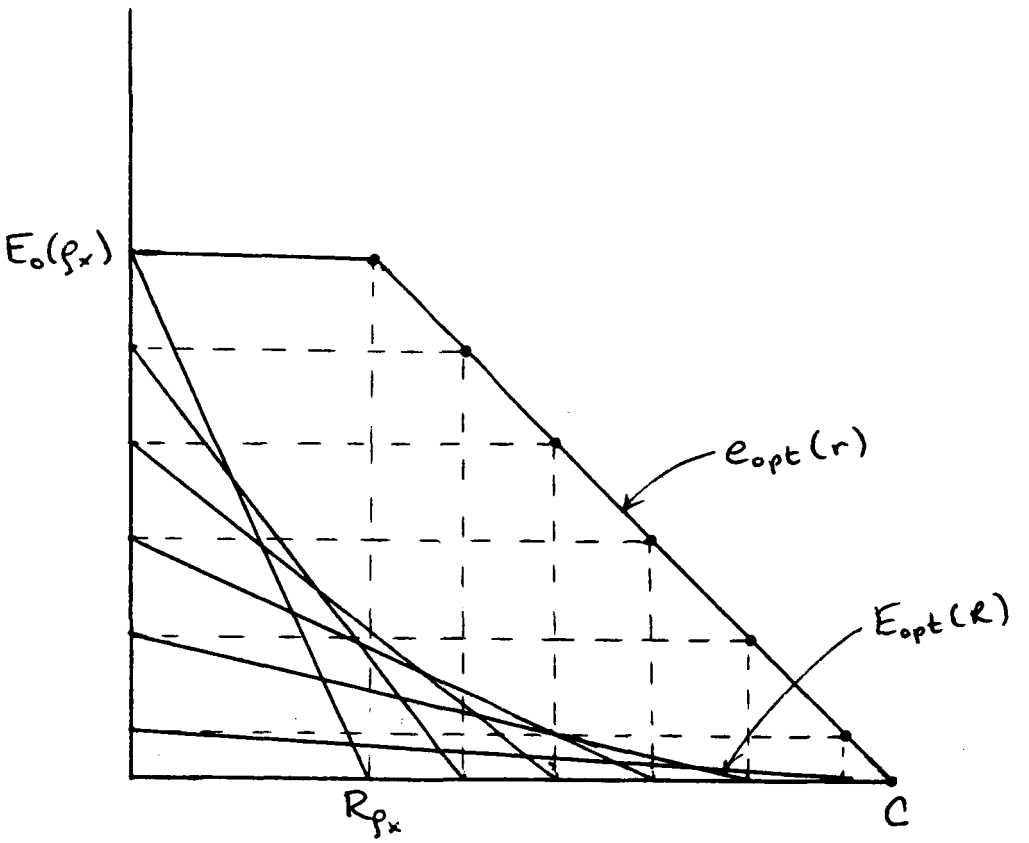


Figure 4. Construction of $e_{opt}(r)$ from $E_{opt}(R)$ Curve

The Merging Concept in the Analysis of Tree Codes

The central concept to be grasped in the analysis of tree codes is that of merging. Two source sequences s and s' are said to be merged at branch i if the two source sequences have the same $\nu + 1$ letters in the positions $i - \nu$ through i:

$$s_{i'} = s'_{i'}, \quad i - \nu \leq i' \leq i. \tag{36}$$

By the definition of constraint length, if s is merged with s' at branch i, the input letters in the branches x_i and x'_i of the corresponding code words must be identical. For example, in a terminated tree code, all source sequences with the same first letter are merged at the first branch, and all with the same last letter are merged at the $(K + \nu)$ th (last) branch.

A corollary concept is that of the unmerged span, which is defined with reference to two particular source sequences, say s and s'. The first unmerged span U_1 contains the indexes of the source letters in the span from the first letter in which s differs from s' to the last letter of the first subsequent string of ν consecutive letters in which s agrees completely with s'. Thus over the first unmerged span s and s' are unmerged, but they are about to merge at the end of the span. The j th unmerged span U_j then contains the indexes of the source letters in the span from the first letter not in U_{j-1} in which s differs from s' to the last letter of the first subsequent string of ν consecutive letters in which s agrees completely with s'. It is obvious that

1. The unmerged spans are disjoint;
2. The unmerged spans contain the indexes of all branches for which s and s' are unmerged.

Thus with the concept of unmerged spans we partition the set of branches at which s and s' are unmerged into disjoint subsets. Each consists of a consecutive string of at least $\nu + 1$ branches; the spans may or may not be separated by merged segments.

The principal utility of this concept lies in the observation that the log likelihood ratio of the code words x and x' corresponding to two

source sequences is equal to the sum of the log likelihood ratios over the unmerged spans. For let y be the received word and y_i, the part of the received word corresponding to the i th branch x_i of the transmitted word; then

$$\ln \frac{\Pr(y|x)}{\Pr(y|x')} = \sum_i \ln \frac{\Pr(y_i|x_i)}{\Pr(y_i|x_i')} = \sum_j \sum_{i \in U_j} \ln \frac{\Pr(y_i|x_i)}{\Pr(y_i|x_i')}, \tag{37}$$

since the branch log likelihood ratio is zero for all i for which s and s' are merged and therefore $x_i = x_i'$.

Now let us suppose that the code word actually sent is x, but that a maximum likelihood decoder chooses some code word $x' \neq x$. From (37) we have immediately

$$\sum_j \sum_{i \in U_j} \ln \frac{\Pr(y_i|x_i)}{\Pr(y_i|x_i')} \leq 1. \tag{38}$$

But the fact that x' has likelihood greater than all other sequences x'' implies the stronger statement

$$\sum_{i \in U_j} \ln \frac{\Pr(y_i|x_i)}{\Pr(y_i|x_i')} \leq 1, \text{ all } j. \tag{39}$$

For suppose the inequality of (39) is not satisfied for some U_j; that is, over the j th unmerged span the correct code word x has greater likelihood than x'. Consider then the source sequence s'' which is equal to s over the span U_j and to s' elsewhere; the corresponding code word x'' will equal x over U_j and x' elsewhere, and will therefore have greater likelihood than x' if (39) is not satisfied.

Consequently, when errors occur with tree codes, it is natural and convenient to think these as separate error events, each one corresponding to a particular unmerged span U_j. Error events may be of any length and contain any number of symbol errors, but do have a definite beginning and end. Intuitively, the number of error events will be proportional to the code length, and one should think not of the probability of error per block, but the probability of error per branch, defined as the probability of an error event starting (or ending) at any particular branch.

When we consider terminated tree codes as block codes, however, we shall be interested in block probability of error. Then the following lemma, which follows directly from the above discussion, will be useful:

Lemma: There will be a decoding error with a terminated tree code if and only if some incorrect word which differs from the correct word only over a single unmerged span has greater likelihood than the correct word.

For suppose the word actually decoded has J unmerged spans with respect to the correct word; by (39) the likelihood ratio over each of these spans will be less than one, so that each of the J words differing from the correct word over a single one of these spans will also have greater likelihood than the correct word.

The merging concept suggests a change in the way we picture tree codes of finite constraint length. Customarily tree codes are depicted graphically as trees, with q^i possible transmitted branches at branch i. Taking into account that sequences do merge with one another, however, we arrive at a structure more like a trellis, with no more than $q^{\nu + 1}$ possible transmitted branches at any one branch. The difference is illustrated in Figure 5, for a binary tree code of constraint length 2 terminated after five information bits. In both the tree and the trellis pictures the representation of any of the 32 particular source sequences of 5 information bits followed by 2 zeroes is the path through the graph labelled by the corresponding bits; in both pictures double lines indicate the representation of 0100000. The code would be totally specified by labelling each branch with the appropriate channel inputs. The trellis picture recognizes that since 0100000 merges with 0000000 after the fourth

Figure 5. Terminated Binary (q=2) Tree Code with ν=2, K=5

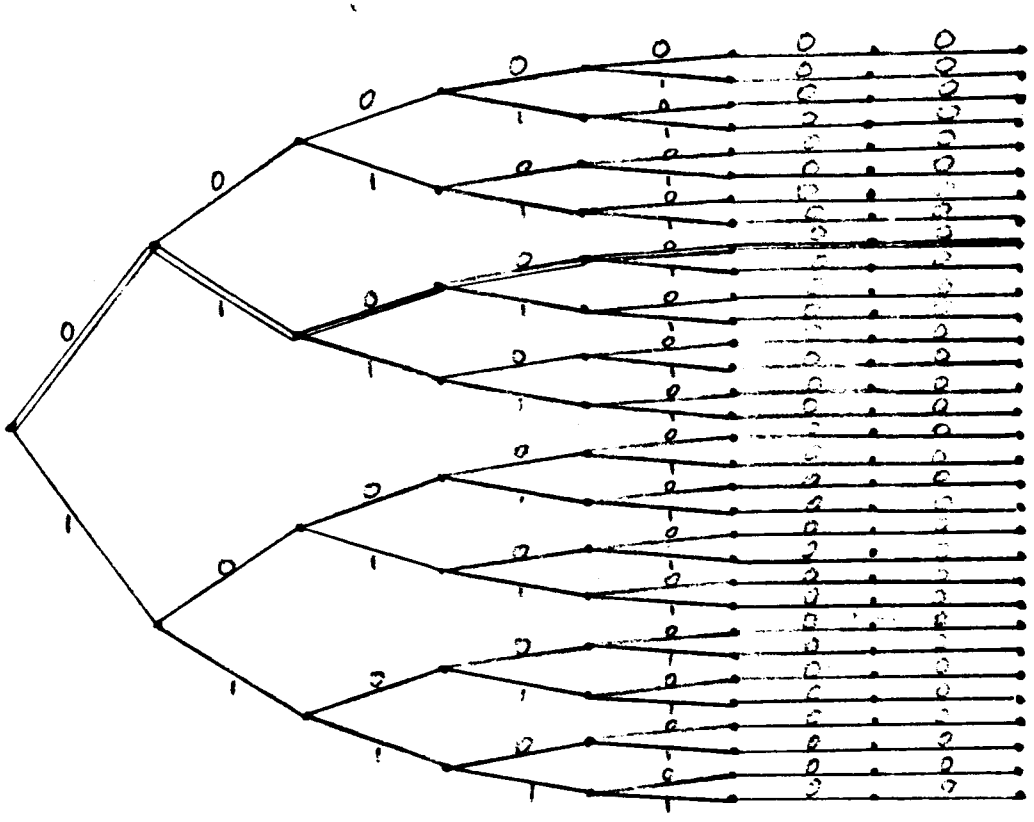


Figure 5a. Tree Picture

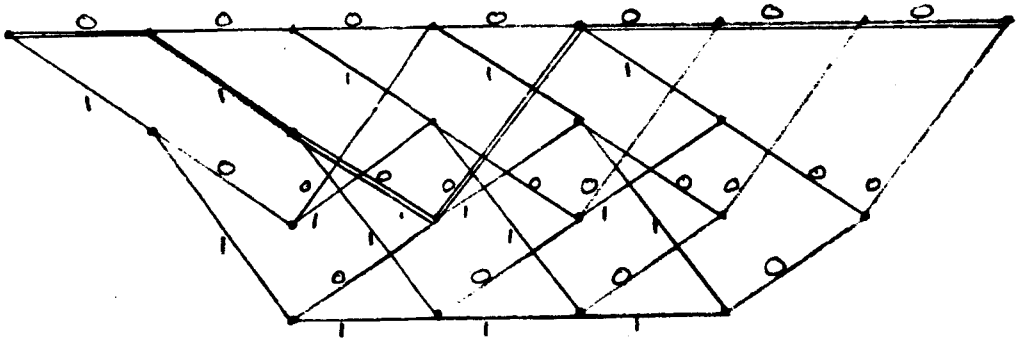


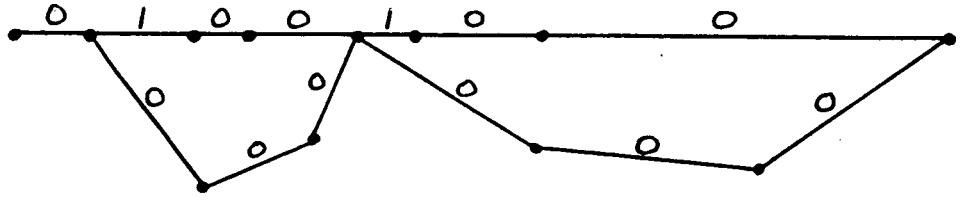
Figure 5b. Trellis Picture (Taking Account of Merging)

branch, the corresponding transmitted branches must be identical for the two code words.

These pictures suggest making up a maximum likelihood decoder out of string, as follows. For each branch, cut out a piece of string of length $- \ln \Pr(y_i \mid x_i)$, where y_i is what was received for branch i and x_i is what would have been transmitted if the actual code word included that branch. For a tree decoder, attach all these pieces of string together as in the tree picture of Figure 5a. Hold the resulting bundle at the tree origin, let all the strings hang down, and pick out the terminal node which is highest; this will be the node corresponding to the shortest path through the tree, thus to the minimum value of

$$- \Sigma \ln \Pr(y_i \mid x_i) = - \ln \Pr(y \mid x) ,$$

and thus to the most likely code sequence. Similarly, for a trellis decoder, attach the pieces of string together according to the trellis topology, as in Figure 5b. Pick up the resulting bundle at the two end points and pull them apart until some path becomes taut; this is again the path of minimum length and hence of maximum likelihood. All less likely paths will dangle down from the most likely path in loops corresponding to unmerged spans. Suppose, for example, that the code word corresponding to 0000000 is actually sent, but that 0100100 is the sequence of greatest likelihood. These two paths alone might look in the string decoder like



The discussion above simply pointed out that the over-all block decoding error ought to be thought of as two distinct error events, one corresponding to each loop, and that the incorrect sequence has to be more likely

than the correct one over each event individually. It is obvious from the picture that both 0100000 and 0000100 are also more likely than 0000000.

Random Tree Codes

A random tree code is one in which the b channel inputs to be assigned to each branch for each of the $q^{\nu+1}$ possible combinations of $\nu+1$ preceding source letters are chosen independently of each other and of all other channel inputs at random according to the input distribution p_k. Thus channel inputs in different positions are totally independent; channel inputs in the same position in two different code words are either identical or totally independent, according to whether the two code words are merged or unmerged at that branch. In the trellis picture, the channel inputs for each distinct branch are picked randomly and independently.

We now compute the block error probability of a terminated tree code with maximum likelihood decoding, which of course remains the optimum decoding technique. By the lemma of the previous section, an error occurs if and only if some code word differing from the correct word only over a single unmerged span has greater likelihood than the correct word. We shall consider separately the sets $S_{ii'}$, defined to consist of all such code words (or the corresponding source sequences) for which the single unmerged span contains branches i to i', $0 \le i \le K$, $i + \nu \le i' \le K + \nu$.

First we determine the size $|S_{ii'}|$ of the set $S_{ii'}$. Since all members terminate in the same ν source letters, there are only $i'-i+1-\nu$ source letters in which they may differ, so that

$$|S_{ii'}| \le q^{i'-i-\nu+1}. \tag{40}$$

On the other hand, the set $S_{ii'}$ certainly contains all words which have a source letter different from that of the correct sequence at the i th position, the $(i'-\nu)$ th position, and at the $(i+j\nu)$ th positions,

$$1 \le j \le \frac{i'-i}{\nu} - 1; \tag{41}$$

therefore at least

$$|S_{ii'}| \geq q^{i'-i+\nu+1}\, q^{-\left[\frac{i'-i}{\nu}-1\right]} . \tag{42}$$

It follows that asymptotically

$$|S_{ii'}| \doteq q^{\nu\left(\frac{i'-i+1}{\nu}-1\right)} . \tag{43}$$

Now all these code words will be pairwise independent of the correct code word over the span from branch i to i', which includes (i'-i+1) b channel inputs. The set $S_{ii'}$, may therefore be thought of as a random block code meeting the three conditions 1, 2', and 3 specified earlier, having length N=(i'-i+1) b, $M = |S_{ii'}|$ code words, and therefore rate R given by

$$\begin{aligned} R &= \frac{\ln M}{N} \\ &\doteq \frac{\nu\left(\frac{i'-i+1}{\nu}-1\right)\ln q}{(i'-i+1)\,b} \\ &= \mu r, \end{aligned} \tag{44}$$

where we have recalled that the tree code rate r = ln q/b, and defined

$$\mu = 1 - \frac{\nu}{i'-i+1} . \tag{45}$$

For such a random block code, we know that the probability of decoding to one of the incorrect words is given by

$$\begin{aligned} P_{ii'}(\mathcal{E}) &\doteq \exp -NE(R) \\ &= \exp -\frac{\nu b}{1-\mu} E(\mu r) \\ &= \exp -\nu b\, e(r,\mu), \end{aligned} \tag{46}$$

where we have defined the tree code exponent $e(r, \mu)$ as

$$e(r, \mu) = \frac{E(\mu r)}{1-\mu} . \tag{47}$$

In Figure 6 we give a graphical construction of $e(r, \mu)$ as the value at R = 0 of a straight line drawn from r through the E(R) curve at the rate $R = \mu r$.

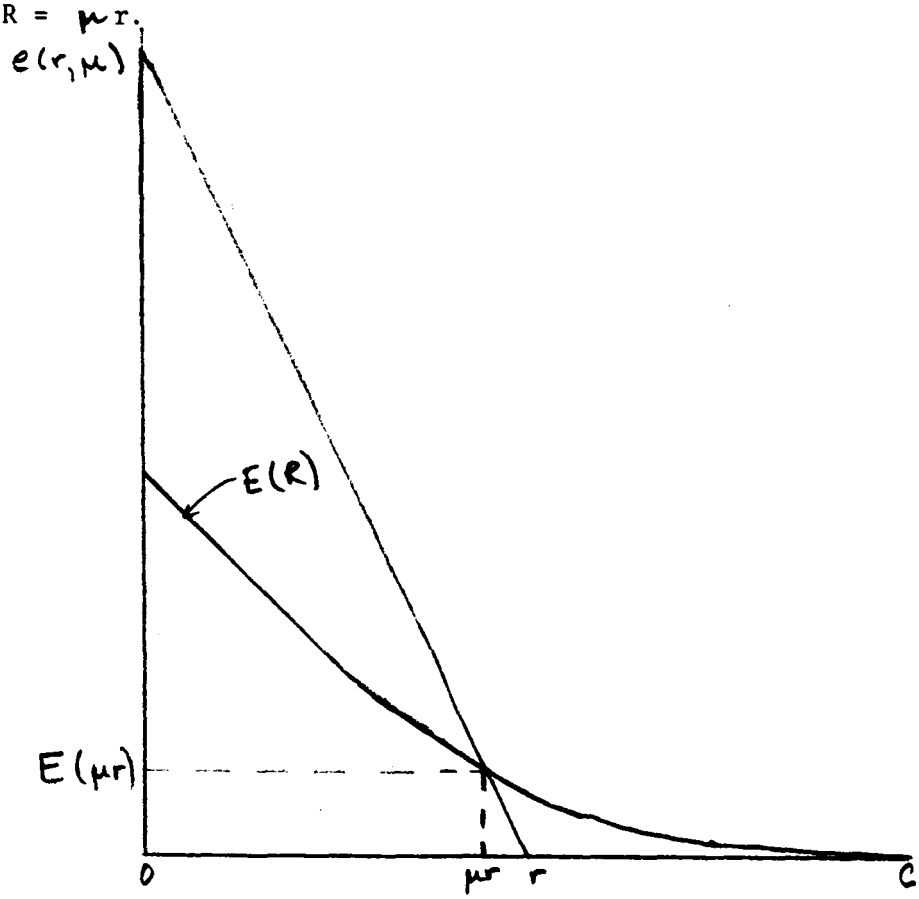


Figure 6. Graphical Construction of $e(r, \mu)$

We now ask for what value of μ Pr_{ii}, (e) is maximum, or $e(r, \mu)$ is minimum. It is obvious from the construction that for $r \leq R_1 = R_{comp}$ the minimum value of $e(r, \mu)$ occurs for $\mu = 0$, where $e(r,0) = E(0) = R_{comp}$, and that for $r \geq R_1$, the minimum occurs when the straight line is tangent to the E(R) curve. We know that if $r = R_\rho$, this tangent has the equation

$$T_{\rho}(R) = E_o(\rho) - \rho R; \tag{48}$$

therefore the minimum value of $e(r, \mu)$ is $E_o(\rho)$. In summary, if we define

$$e(r) = \min_{0 \le \mu \le 1} e(r, \mu), \tag{49}$$

then for low rates

$$e(r) = E(0) = R_{comp}, \; r \le R_1 = R_{comp}; \tag{50}$$

and for high rates, we have

$$e(r) = E_o(\rho), \quad r = R_{\rho}, \; r \ge R_1. \tag{51}$$

We immediately recognize this latter expression as equal to that of (35) for $e_{opt}(r)$ when $r \ge R_1$.

Now let us show that $e(r)$ is actually the random tree code error exponent. The over-all probability of decoding error for a terminated tree code is lower-bounded by the probability of decoding incorrectly within any set $S_{ii'}$, and upper-bounded by the sum of the probabilities of decoding incorrectly in all sets, by the union bound:

$$P_{r_{ii'}}(\mathcal{E}) \le P_r(\mathcal{E}) \le \sum_{i=1}^{K} \sum_{i'=i+\nu}^{K+\nu} P_{r_{ii'}}(\mathcal{E}). \tag{52}$$

But from the discussion just previous, the largest value of $P_{r_{ii'}}(\mathcal{E})$ is asymptotically equal to

$$P_{r_{ii'}}(\mathcal{E})_{max} \doteq \exp - \nu be(r), \tag{53}$$

and if K is large enough there exists an i and i' such that this maximum is achieved. Hence we can say

$$Pr_{ii'}(\mathcal{E})_{max} \leq Pr(\mathcal{E}) \leq K^2 Pr_{ii'}(\mathcal{E})_{max}, \tag{54}$$

and if K increases less than exponentially with ν,

$$Pr(\mathcal{E}) \doteq \exp - \nu be(r) \tag{55}$$

for a terminated tree code. For a nonterminated tree code, (55) continues to apply if we define $Pr(\mathcal{E})$ as the probability of error per branch, as can be seen by repeating the above argument with i or i' fixed.

Another suggestive expression for the tree code error probability comes from substituting for b in (55) to obtain

$$Pr(\mathcal{E}) \doteq q^{-\nu e(r)/r} \tag{56}$$

Letting $\rho(r)$ be the ρ for which $r = R_\rho$, and recalling that $e(r) = E_o[\rho(r)]$ for $r \geq R_1$, (56) becomes

$$Pr(\mathcal{E}) \doteq q^{-\nu \rho(r)}, \quad r \geq R_1. \tag{57}$$

At lower rates $e(r) = R_1 = R_{comp}$ and (56) becomes

$$Pr(\mathcal{E}) \doteq q^{-\nu R_1/r}, \quad r \leq R_1. \tag{58}$$

Consequently, for all rates the random tree code exponent is known exactly and given by (50) and (51), just as the random block code exponent is known for all rates. Further, it is clear that if the terminated random tree code is considered as a block code, then by the construction of Figure 3 the equivalent block code exponent obtainable is, using (50) and (51),

$$E(R) = \max_{0 \leq \rho \leq 1} [E_o(\rho) - \rho R] \tag{59}$$

which is precisely the random block code exponent. Third, for $r \geq R_1 = R_{comp}$, the random tree code exponent is equal to the optimum tree code exponent $e_{opt}(r)$ given by (33), just as for $R \geq R(1) = R_{crit}$ the random block code exponent is equal to the optimum block code exponent; thus at high rates random tree codes are a solution to the tree code construction problem. Finally, when considered as a block code, and with proper choice of the synchronization rate loss λ, the terminated random tree code has a block code exponent equal to the optimum block code exponent for block code rates $R \geq R(1)$; thus at high rates random tree codes are a solution to the block code construction problem.

In conclusion, with proper choices of parameters, terminated random tree codes are just as good in performance as random block codes, and are therefore optimum block codes wherever the latter are optimum. However, so far we have assumed maximum likelihood decoding, so that decoding complexity is presumably also exactly equal to that for random block codes, so it is not yet clear whether tree codes offer any advantages. Subsequently we shall see that their trellis structure permits a decoding algorithm much less complex than that for block codes, so that when considered as block codes they have a complexity advantage. This decoding algorithm will suggest, however, that tree codes are better <u>not</u> terminated, and when used in this way tree codes will be shown to have a performance advantage as well. Before we proceed with these principal themes, however, we pause briefly to consider the character of error events with random tree codes, as is natural at this point.

<u>Character of Error Patterns with Random Tree Codes</u>

We have seen that the probability of error within unmerged spans $S_{ii'}$ of length $\nu/(1-\mu)$ is given by

$$Pr_{\mu}(\mathcal{E}) \doteq \exp - \nu\, b[E(\mu r)/(1-\mu)] . \qquad (60)$$

Such an error results in an error sequence of length $\nu/(1-\mu) - \nu = \nu\mu/(1-\mu)$, which is equally likely to be any of the unmerged sequences of that length. If the random tree code is very long, then, we would expect to see error sequences of length $\nu\mu/(1-\mu)$ at about the relative

frequency per branch given by (60).

The construction of Figure 6 determined the μ_o for which the exponent in (60) was minimum. Error sequences of length near $\nu\mu_o/(1-\mu_o)$ are therefore going to occur most frequently in the decoded output; because of the exponential character of (60), error sequences of substantially different μ are going to occur with negligible frequency for large constraint lengths ν. We can say a little bit about the range of μ which is interesting. Define error sequences of length $\nu\mu/(1-\mu)$ as rare if

$$\frac{Pr_\mu(\mathcal{E})}{Pr_{\mu_o}(\mathcal{E})} \leq A, \tag{61}$$

where A is any constant; this is equivalent, from (60), to

$$\frac{\nu b E(\mu r)}{(1-\mu)} - \frac{\nu b E(\mu_o r)}{(1-\mu_o)} \geq \ln A. \tag{62}$$

Let

$$f(\mu) = \frac{E(\mu r)}{(1-\mu)r}; \tag{63}$$

then in terms of a new constant B, (62) can be expressed

$$f(\mu) - f(\mu_o) \geq B/\nu. \tag{64}$$

Let us then inquire into what happens when ν becomes large. We can expand $f(\mu)$ in a Taylor series about μ_o to obtain

$$f(\mu) = f(\mu_o) + (\mu-\mu_o)f'(\mu_o) + \frac{(\mu-\mu_o)^2}{2} f''(\mu_o) + \cdots \tag{65}$$

But at μ_o the first derivative of $f(\mu)$ is zero:

$$f'(\mu_o) = \frac{rE'(\mu_o r)}{(1-\mu_o)r} + \frac{E(\mu_o r)}{(1-\mu_o)^2 r}$$

$$= -\frac{\rho r}{(1-\mu_o)r} + \frac{(1-\mu_o)r\rho}{(1-\mu_o)^2 r} \qquad (66)$$

$$= 0,$$

where we have used the fact that μ_o is chosen so that $E(\mu_o r) = (1-\mu_o)r\rho$, $-\rho$ being the slope of the line from r to $E(\mu r)$. Substituting (66) and (65) into (64) yields

$$(\mu-\mu_o)^2 \geq \frac{2B}{\nu}, \qquad (67)$$

where we suppose ν is large enough that higher-order terms in (60) can be neglected. The range of μ which satisfy (67) is therefore proportional to $\nu^{-1/2}$; the range of error sequences which are not rare is consequently proportional to $\nu^{1/2}$. As ν increases, therefore, most error sequences tend to have lengths near $\nu\mu_o/(1-\mu_o)$, although the absolute range increases, in the same way as the sum of a large number N of random variables clusters more closely about the N times the mean, though the dispersion is increasing proportional to $\sqrt{N}$.

For a tree code of rate $r = R_\rho \geq R_1$,

$$\mu_o = \frac{E_o'(\rho)}{R_\rho}, \qquad (68)$$

so that the typical length of error sequences is

$$\nu \frac{E_o'(\rho)}{R_\rho - E_o'(\rho)}. \qquad (69)$$

In particular, if the tree code rate is $R_1 = R_{comp}$, then typically error sequences will be of length

$$\nu \frac{R_{crit}}{R_{comp} - R_{crit}}, \tag{70}$$

where $R_{crit} = E_o'(1)$.

The Viterbi Algorithm

We now show that the structure of tree codes permits a decoding algorithm which gives precisely the same results as maximum likelihood decoding and is therefore optimum, but which has complexity proportional only to q^{ν}, rather than to the q^K required by maximum likelihood decoding of a terminated tree code. That something like this is possible was foretold by our string decoders, where the number of pieces of string required was proportional to q^K for the tree structure, but only to q^{ν} for the trellis.

The critical observation to be made is the following. Let s and s' be two input words which agree for the span of ν consecutive letters ending in i; that is, which are either merged at branch i or which have an unmerged span terminating at i. Let s be the one which has greater likelihood up to branch i:

$$\sum_{i'=1}^{i} \ln \frac{\Pr(y_{i'} | x_{i'})}{\Pr(y_{i'} | x'_{i'})} \geq 1. \tag{71}$$

Then s' cannot be the choice of a maximum likelihood decoder. For s' will always have less likelihood than the input sequence which agrees with s up to branch i and with s' thereafter, since the corresponding code word will be identical to x' beyond branch i, and from (71) will therefore continue to have greater likelihood forever.

Let us rephrase this observation in terms of our trellis structure, and our trellis string decoder. Suppose we set up the structure only out to branch i, and pulled on one of the q^{ν} nodes at that depth and

the initial node, thus finding the shortest path out to that node. Clearly any other path dangling down in a loop from the shortest path is not going to be a part of the path finally chosen; thus we can take out scissors and cut away any such loops without any possibility of discarding the most possible path. If we do this for all q^{ν} nodes, we shall be left with only one path from the initial node to each of these q^{ν} nodes, which set will now form a true tree, with many branches missing, but no loops. This operation can be repeated for each branch in sequence, until one comes to the final node, at which time only one path will remain, necessarily the shortest one of all and thus the optimum choice of a maximum likelihood decoder. This is the Viterbi algorithm.

Returning to the world where decoders are built with computer components rather than string, let us restate the algorithm. The decoder operates in a series of steps, one for each branch past branch ν. At the step associated with branch i, for each of the q^{ν} possible strings of source letters of length ν, it chooses the one input sequence which, among all those which have that particular string of ν letters ending at branch i, has the greatest likelihood up to branch i. At the end of any step, therefore, it retains a list of only q^{ν} possible sequences up to branch i; this determines the size of the decoder memory. It follows that at the next step, there will be only q sequences to be compared in each class, so only $q^{\nu + 1}$ likelihoods to be computed in all. It is therefore evident that the complexity of the algorithm is indeed asymptotically proportional to q^{ν}, if the length of the sequences does not grow exponentially with ν.

This algorithm may never actually make a final choice until the tree code is terminated, although a choice can be imposed after a while without any loss in performance, as we shall discuss later. Should the tree code be terminated, however, the algorithm automatically converges on a single choice, since it need not consider words which do not agree at the end with the ν-letter synchronizing sequence, and therefore at the step associated with the branch $K + \nu$ it chooses the one input sequence which, among all those ending with the synchronizing sequence, has the greatest likelihood over the whole block. This sequence must be the same sequence

as would be chosen by a maximum likelihood decoder, since the only words ever discarded are those which could not possibly be the choice of a maximum likelihood decoder. The algorithm is therefore optimum--Viterbi too modestly called it suboptimum but asymptotically optimum--and consequently we know the error probability which can be obtained with its use.

Expressions which give the probability of error $\Pr(\mathcal{E})$ and decoding complexity G when random tree codes are used with the Viterbi algorithm are therefore

$$P_r(\mathcal{E}) \doteq \begin{cases} q^{-N\rho(r)}, & r \geq R_1; \\ q^{-NR_1/r}, & r \leq R_1; \end{cases}$$

$$G \doteq q^N, \tag{72}$$

where we have reproduced (57) and (58). The probability of error is therefore given directly in terms of the decoding complexity by

$$P_r(\mathcal{E}) = \begin{cases} G^{-\rho(r)}, & r \geq R_1; \\ G^{-R_1/r}, & r \leq R_1. \end{cases} \tag{73}$$

Again the probability of error decreases only algebraically with complexity, as with maximum likelihood decoding of block codes. However, because the complexity is less with tree codes, the exponent is greater. For a direct comparison of exponents, compare a block code of rate R, error exponent E(R), and therefore [from (22)] complexity exponent E(R)/R, with the terminated tree code of the right tree code rate r_R and the right λ to give an optimum block code of rate R [that is, if $R=E_o'(\rho)$, $r_R=R_\rho$], which will have complexity exponent $\rho(r_R)$; this comparison is made graphically in Figure 7. Even with the allowance for synchronization loss, the comparison strongly favors the tree code, the more so the closer the rate approaches capacity. Obviously the comparison would be even more favorable to the tree code if we let the block length increase indefinitely

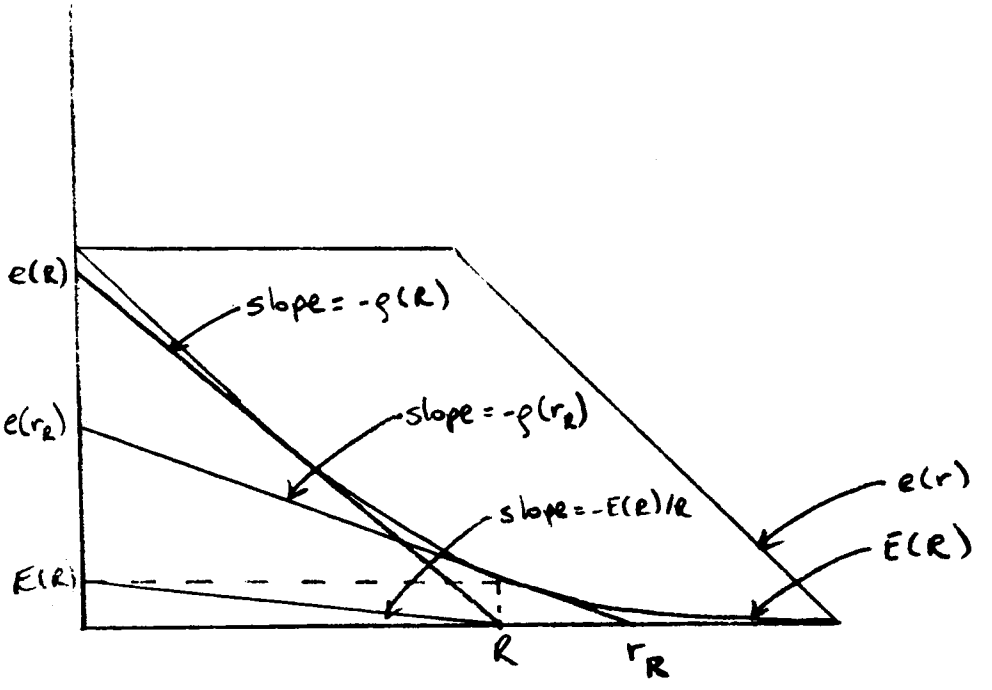


Figure 7. Comparison of Block and Tree Code Exponents

so that r approached the block code rate R, since the complexity and probability of error depend only on ν and r, and not on the block length; then we would get an exponent equal to $\rho(R)$, the ρ for which $R=R_\rho$, as is also illustrated in Figure 7. Again, we are led to feel that although random tree codes can be used as optimum block codes, it is better not to terminate them, or at least to let the blocks between synchronizing sequences become very long. Though it is true that in this case the equivalent block code exponents go to zero, one is usually more interested in probability of error versus complexity than versus block length.

Mismatched Decoding Constraint Length

Let us now suppose that the decoder uses the Viterbi algorithm appropriate for a code of constraint length ν, but that actually the encoder constraint length is some ν' not equal to ν. What will happen?

First, suppose ν' is less than ν. A code of encoding constraint length ν' may be considered as a poor example of a code of encoding constraint length ν, if $\nu' \leq \nu$, for indeed each branch depends on only the $\nu + 1$ preceding input letters. Thus the Viterbi algorithm remains an optimum maximum likelihood decoder for the shorter code; the mismatch results only in increasing the decoding effort to q^{ν}, rather than the $q^{\nu'}$ which would be sufficient.

Second, suppose ν' is actually greater than ν. Consider first the case in which the decoder arrives at branch i' having made no previous decoding error, which is to say that among the q^{ν} retained code sequences one is actually correct. The preceding analyses, which applied when the encoding constraint length was ν, showed that an error occurs at branch i' if and only if over an unmerged span which terminates at i' an incorrect word has greater likelihood than the correct sequence--that is, if and only if among the set of words which agree with the correct word in the last ν places, there is one better than the correct one, so that the algorithm will choose it. But the situation is identical if the encoding constraint length is actually greater than ν but no previous error has been made, for an error is still made if and only if among the set of words which agree with the correct word over the last ν places,

one is better than the correct one, and each word in this set is identical to the correct word up to the beginning of the unmerged span and independent of it thereafter, just as before. It follows that the probability of error, given no previous decoding error, remains

$$\Pr(\mathcal{E}) \doteq \exp - \nu\, b\, e(r), \tag{74}$$

where ν is the decoding constraint length. But then the block probability of decoding error is also given by (74), since it is no greater than K times the branch probability. In sum, when ν' and ν are the encoding and decoding constraint lengths,

$$\Pr(\mathcal{E}) = \exp - \nu''\, be(r),$$
$$\nu'' = \min(\nu, \nu'). \tag{75}$$

What does happen after a decoding error? Until by some chance one of the q^{ν} survivors at some step agrees with the correct word in the last ν' positions, all further choices will be between sets of incorrect words, so that all subsequent likelihoods would be expected to be small on the average. Even when by some chance a survivor does truly merge with the correct word, it may very well succumb in later comparisons, since over its unmerged span it may have accumulated a very poor likelihood. Only when at some subsequent point are all survivors simultaneously remerged with the correct path at some branch is the decoding process truly resynchronized.

A Sequential Viterbi Algorithm

The considerations of the previous section lead us to propose the following algorithm, suitable for a code of very large, essentially infinite encoding constraint length. We rely on the fact that in this case, if a decoding error is ever made, it will sooner or later become obvious, and any sensible criterion will detect the occurrence of the error. Then begin by decoding with a Viterbi decoder suitable for some small constraint length ν_0. If an error is ever detected, begin again with a

Viterbi decoder of some larger decoding constraint length ν_1. Continue in this way through a series of increasing constraint lengths until the whole block is decoded. Assuming perfect detection, the probability of error will be zero. Assuming that the length of the block is not exponentially large, the probability of having to invoke a decoder of decoding constraint length greater than ν_i and hence complexity $G \doteq q^{\nu_i}$ is the probability of decoding error with decoding constraint length ν_i, or, from (56),

$$\Pr\left(G \doteq q^{\nu_i}\right) \doteq q^{-\nu_i\, e(r)/r} \tag{76}$$

or, if $r \geq R_1 = R_{comp}$, from (57),

$$\Pr\left(G \doteq L\right) \doteq L^{-\rho(r)}, \quad r \geq R_{comp}, \tag{77}$$

where $\rho(r)$ is the ρ for which $r = R_\rho$. This is precisely the behavior characteristic of sequential decoding for rates greater than the computational cutoff rate; the exponents are identical. By thus exhibiting an algorithm which is so seemingly different from the sequential decoding algorithms, yet which nonetheless has the same distribution of decoding complexity, we further confirm the observation of Jacobs and Berlekamp (1967) that such a distribution is characteristic of optimum sequential tree code algorithms generally and not just of sequential decoding. Of course, this particular algorithm is not practically attractive.

Below $R_1 = R_{comp}$, our present results enable us to say only that

$$\Pr\left(G \doteq L\right) = L^{-R_1/r}, \quad r \leq R_{comp}. \tag{78}$$

This is sufficient to show that below the computational cutoff rate the exponent exceeds 1, but is not as strong as the result known for sequential decoding, namely that the exponent continues to equal $\rho(r)$, at least at rates $r = R_{\rho(r)}$ where $\rho(r)$ is an integer. We conjecture that the Viterbi algorithm could be modified to yield this performance by retaining at each step of the algorithm, among each set of words having

identical source letters in the last ν branches, the L with the greatest likelihoods, rather than just the single best. However, the proof of this requires a list decoding result for tree codes, which relates to the block code list result by the construction of Figure 3; such a result can be proved if we ignore the fact that incorrect sequences may merge among themselves, but so far not otherwise.

These results suggest that the event in which the decoding complexity (computation) required is greater than q^{ν} is identical to the event in which a decoding error would be made if the encoding constraint length were indeed only ν . In the next sections we discuss some list decoding results of Jacobs and Berlekamp which qualitatively support this supposition, and draw a moral for sequential decoding.

Results of Jacobs and Berlekamp

Jacobs and Berlekamp show that no sequential algorithm can have a complexity exponent better than $\rho(r)$ by making the following argument:

1. Let a sequential algorithm be defined as one which examines the possible code sequences up to branch i in an order which does not depend on received branches beyond i, and let its complexity be at least as great as the number of sequences so examined before coming to the correct sequence.

2. An algorithm which examines sequences in order of their likelihoods will on the average examine fewer sequences before coming to the correct sequence than an algorithm which examines sequences in any other order.

3. The probability that the complexity will exceed L is therefore at least as great as the probability that the correct word will not be among the words with the L greatest likelihoods, or thus that an error would be made with list-of-L decoding.

4. The truncations of all possible code sequences to branch i form a block code of length N=ib, with $M=q^i$ different words, and therefore with rate R given by

$$\begin{aligned} R &= \frac{\ln M}{N} \\ &= \frac{\ln q}{b} \\ &= r, \end{aligned} \tag{79}$$

the tree code rate.

5. The probability of list decoding error with a block code of rate r and a decoding list of size

$$L \doteq \exp\,(1-\mu)\,Nr,\quad 0 \le \mu \le 1, \tag{80}$$

is bounded by

$$\Pr_{\mu}(\mathcal{E}) \doteq\!\geq \exp - NE_{sp}(\mu r), \tag{81}$$

from (17).

6. Thus, substituting (80) into (81),

$$\Pr(C > L) \doteq\!\geq L^{-\frac{E_{sp}(\mu r)}{(1-\mu)r}}$$

$$= L^{-\rho(r,\mu)}. \tag{82}$$

Geometrically, $-\rho(r,\mu)$ is the slope of the straight line which equals 0 at $R = r$ and $E_{sp}(\mu r)$ at $R = \mu r$, as is illustrated in Figure 8; clearly this is precisely the construction of Figure 6, which was used to find the tree code exponent. As there, the minimum magnitude of the slope comes when μ is such that the straight line is tangent to the sphere-packing exponent; for this μ, earlier called μ_o, we have

$$\rho(r,\mu_o) = \rho(r), \tag{83}$$

where $\rho(r)$ is as always the ρ for which $r=R_\rho$. Choosing $\mu = \mu_o$, we have

$$\Pr(C > L) \doteq\!\geq L^{-\rho(r)} \tag{84}$$

for any sequential decoding algorithm with complexity C as defined above.

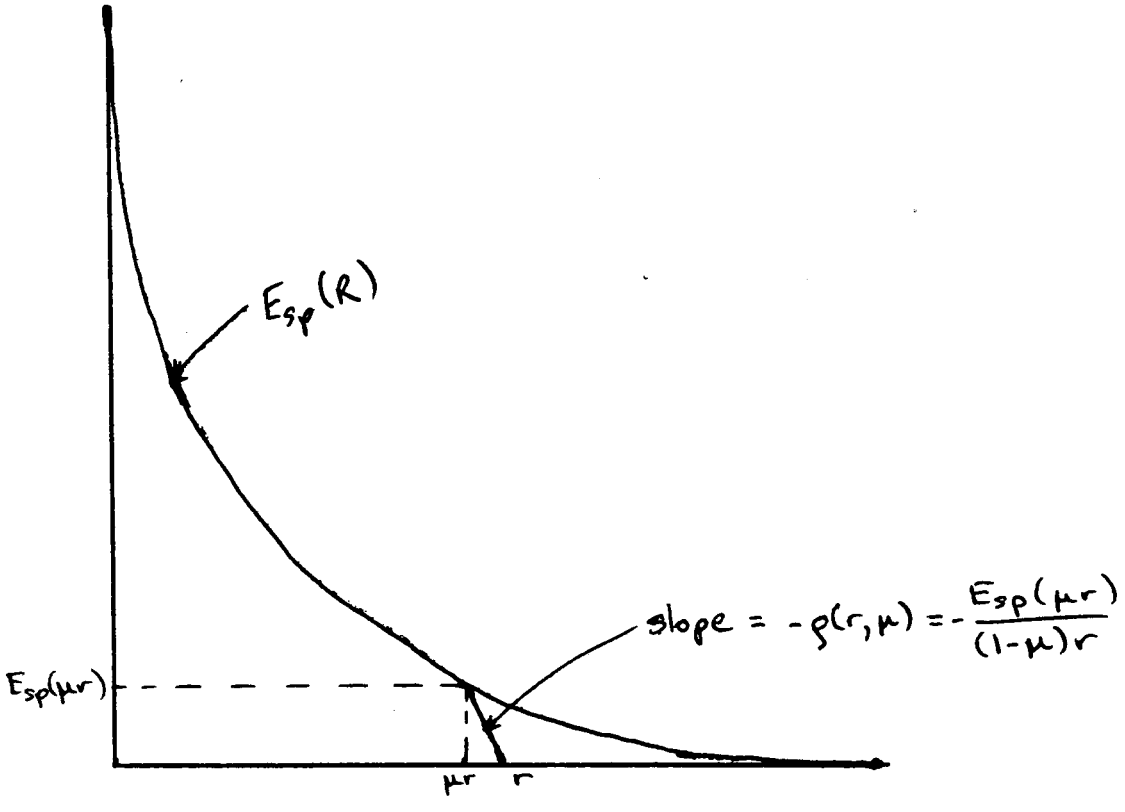


Figure 8. Construction of $\rho(r,\mu)$.

The similarity of the quantities arising in this development to those in our earlier development of the probability of error for tree codes suggests a basic kinship between the two situations. We feel it is of value to elucidate this kinship.

First, let us rework the result of Jacobs and Berlekamp. As it stands, it has some peculiar features: For a fixed value of L, for example, the bound on $\Pr(C > L)$ increases up to a certain critical length N_o for which $L \doteq \exp(1-\mu_o)N_o r$, and then decreases subsequently, suggesting that once the decoder gets over an initial lump in computation, it is in the clear. This particular quirk can be fixed for $N \geq N_o$ by considering the block code consisting of the words which are identical to the correct word up to branch $(N-N_o)/b$, in effect "starting" the tree code sufficiently later so that its block code length is always the critical length N_o, and then we obtain $\Pr(C > L) \geq L^{-\rho(r)}$ for all L and all $N \geq N_o$. This strongly suggests that the JB result is really a "per branch" rather than a "per block" result.

For an infinite constraint length code, define $S_{ii'}$ as the set of all sequences of i' branches which first unmerge at branch i. It is not hard to show that the size of this set is asymptotically equal to

$$|S_{ii'}| \doteq q^{i'-i+1}; \tag{85}$$

the argument is similar to that leading to (43). The code words corresponding to these sequences form a block code of effective length $N = (i'-i+1)b$, since we may ignore the places before the i th branch, and the rate of the block code, from (85), is therefore approximately the tree code rate r. Now define $L_{ii'}$ as the number of elements of the set $S_{ii'}$ which have greater likelihoods than the correct code word, and define μ by

$$L \doteq \exp(1-\mu)Nr. \tag{86}$$

Then by the list decoding lower bound

$$\Pr(L_{ii'} > L) \doteq \exp - NE_{sp}(\mu r)$$
$$= L^{-} \varphi(r,\mu) \tag{87}$$

where we simply repeat the arguments of (80) - (82). The JB result then follows from

$$\Pr(C > L) \geq \Pr(L_{ii'} > L), \text{ any } i, i'$$
$$\doteq L^{-\min_{\mu}} \varphi(r,\mu) \tag{88}$$
$$= L^{-} \varphi(r).$$

But what we want to note is that the bound of (88) is tight at high rates. For let us set up a particular code, namely a random tree code, and a particular list-of-L decoding scheme, namely a Viterbi decoder of decoding constraint length ν, where $L=q^{\nu}$; the list generated by such a decoder is to be considered as the q^{ν} survivors of the decoding step at branch i'. Clearly the correct word is not on this list only if it is discarded within the group of sequences which agree with it in the last ν letters, and we have determined that the probability of this event is

$$\Pr(\mathcal{E}) \doteq \exp - \nu be(r)$$
$$= \exp - \nu br \varphi(r), \; r \geq R_1, \tag{89}$$
$$= L^{-} \varphi(r),$$

which agrees with (88) for $r \geq R_1 = R_{comp}$. Thus we see that at high rates not only is the bound of (88) tight, but that to asymptotic precision the event in which the computations exceed q^{ν} with an infinite constraint length sequential decoder is precisely the event in which a Viterbi decoder of decoding constraint length ν would make an error. From this we shall in the next section draw a moral.

For completeness, we now extend the above argument to show that (87) is tight as well, at high rates. Consider a modification of the above

scheme, also using a Viterbi-like decoder of constraint length ν , where again $L \doteq q^{\nu}$. The decoder cannot know which words belong to the set $S_{ii'}$, or which is the correct word, but it can do the following. Within each of the q^{ν} subsets of sequences which agree with each other over the consecutive letters ending with branch i, it can set up an ordered list of the input sequences in order of likelihood. It can then submit these to a judge, with instructions to construct a list of L by taking the topmost member of each list which actually either belongs to $S_{ii'}$, or is the correct word; the judge then determines whether this list of $L=q^{\nu}$ sequences is totally in $S_{ii'}$, or contains the correct word. The probability of the correct word not being on the judge's list is then the probability that some member of the set $S_{ii'}$, which agrees with the correct word over the last ν consecutive letters is more probable than the correct word, which we determined in (46) to be equal to

$$Pr_{ii'}(\mathcal{E}) \doteq \exp - \nu b\, e(r, \mu) \tag{90}$$

where

$$\mu = 1 - \frac{\nu}{i'-i+1}, \tag{91}$$

from (45), and

$$e(r, \mu) = r\, \rho(r, \mu), \tag{92}$$

from comparison of Figures 6 and 8. Thus (90) can be written

$$Pr_{ii'}(\mathcal{E}) \doteq \exp - \nu br\, \rho(r, \mu) \tag{93}$$

$$= L^{-\rho(r,\mu)},$$

which agrees with (87). Thus we see by examination of this particular scheme that for $\mu r \geq R_{crit}$ the probability that the number $L_{ii'}$ of elements of $S_{ii'}$ which have greater likelihoods than the correct code word exceeds L is asymptotically equal to the expression of (93).

As in our discussion of the sequential Viterbi algorithm, we conjecture that these bounds could be shown to be tight at all rates by considering a Viterbi-like decoder which within each group chose a small list of words, rather than a single survivor.

Morals for Sequential Decoding

The standard sequential decoder design uses a code of constraint length long enough that decoding errors are negligible; the probability of decoding failure is then dominated by the probability that decoding computation becomes excessive. We would suggest that, unless the ability to detect decoding errors is highly prized, one might do better to reduce the constraint length to the point that decoding errors predominate. Our reasoning follows.

First, let us use the results of the previous section to analyze the tradeoff that could be made. A conventional sequential decoder, say of the Fano type, is characterized by the maximum number of computations L that can be made on one branch, or typically by the product of the buffer size in branches times the number of computations which can be made in the time taken to transmit one branch. If this latter speed factor is substantially greater than the average decoding load per branch, the probability of decoding failure is approximately given by [Jordan (1966)]

$$\Pr(\mathcal{E}) \doteq L^{-\rho(r)} \tag{94}$$

The results of the previous section suggest that if we shorten the encoding constraint length to approximately ν, where $q^{\nu} \doteq L$, this probability of error will be unaffected, but errors will now tend to be decoding errors rather than buffer overflows. (With a Viterbi-type sequential decoder this correspondence would be exact; with other sequential decoders, only approximate.) Thus we have some freedom to make one or the other type of error predominate without grossly affecting the error probability.

Now which type of error is more desirable? The buffer overflow type has the great virtue that it is always detectable. It has the great

disadvantage, however, that after overflow one must wait to become resynchronized to resume decoding; this may involve waiting for the block termination with terminated tree codes, or perhaps automatic resynchronization, as will be discussed later. In any case resynchronization requires special procedures and involves a dropout of many bits, typically many constraint lengths. With a decoding error, on the other hand, the decoder simply blunders on normally, no special procedures are required, and, as we have seen earlier, error runs are typically of the order of a few constraint lengths or less. If minimizing output error probability and decoder complexity are the principal criteria, decoding errors are therefore to be preferred.

One disadvantage of the conventional sequential decoding algorithms vis-a-vis the Viterbi-type is that buffer overflow cannot be eliminated. With an ideal Viterbi-type sequential decoder we might have the computational distribution

$$\begin{aligned} \Pr(C > L) &\doteq L^{-\rho(r)}, \quad L < q^{\nu} \quad ; \\ &= 0 \qquad\quad , \quad L \geq q^{\nu} \quad . \end{aligned} \tag{95}$$

Such a distribution has a mean given approximately by

$$\bar{C} \doteq \begin{cases} \dfrac{\rho(r)}{\rho(r)-1}, & \rho(r) < 1; \\ \nu \ln q, & \rho(r)=1 \ (r=R_1=R_{comp}); \\ q^{\nu[1-\rho(r)]}, & \rho(r) > 1. \end{cases} \tag{96}$$

If the average number of computations per branch were substantially greater than this mean, then one could be fairly sure that, if the buffer were large enough to permit somewhat more than q^{ν} computations maximum on any one branch, it would never overflow, so that one could indeed omit internal resynchronization procedures in the decoder, relying perhaps on manual intervention. However, the conventional sequential decoding

algorithms, such as the Fano, not only have some variability on how many microcomputations go into a "computation," or a path rejection, but also differ fundamentally from the Viterbi algorithm in not ignoring paths which have merged with some better path. We may say approximately that such algorithms examine at any one branch all sequences in order of likelihood until arriving at either the correct sequence or some sequence merging with the correct sequence, for either one will look good thereafter, while any other will tend to look bad. Arguing purely heuristically, then, we may estimate the distribution of computation for a sequential decoder with a finite-length code as follows. If the code had infinite constraint length, then the probability that L or more words would have greater likelihood than the correct word would be

$$\Pr(C \geq L) \doteq L^{-\rho(r)}. \tag{97}$$

Let the constraint length actually be ν ; we have seen that this does not affect decoder operation until paths remerge. Of the L sequences with greater likelihood than the correct word, each one independently has probability $q^{-\nu}$ of terminating in the same ν letters as the correct sequence, so the probability that none are merging with the correct word is

$$(1 - q^{-\nu})^{L}, \tag{98}$$

which expression is bounded and approximated by

$$(1 - q^{-\nu})^{L} \doteq \exp - Lq^{-\nu}. \tag{99}$$

Thus for $L < q^{\nu}$, this factor is nearly 1, while for $L > q^{\nu}$, it begins to decrease exponentially with L. Taking C' as the number of words with greater likelihood than the most likely sequence merging with the correct word, we have

$$\Pr(C' > L) \doteq (1-q^{-\nu})^{L} L^{-\alpha(r)} \tag{100}$$

as an approximate expression for the computational distribution to be expected with the Fano algorithm and a finite constraint length. This expression seems to differ fundamentally from (95) in not imposing a strict ceiling to the number of computations possible. However, it does become small rapidly for $L > q^{\nu}$, and in fact has approximately the same means as (95), given in (96), which suggests that in practice the probability of buffer overflow can indeed be made negligible without great reduction of ν and thus increase in $Pr(\mathcal{E})$. This conjecture remains to be experimentally verified.

Unterminated Tree Codes

Aside from dividing code sequences into blocks and thus giving the decoder an easy way to start or restart, we have so far discovered no advantage to terminating tree codes. Moreover, termination has the disadvantage of introducing a small rate loss from the tree code rate; this may be more or less objectionable in itself, and also implies in equipment a more or less inconvenient buffering and clocking problem. We are therefore motivated to ask whether we could dispense with termination altogether.

Before treating the resynchronization problem, we first deal with a more trivial one. The Viterbi algorithm never actually makes a final choice on any source letter, as we described it; without termination, its q^{ν} surviving sequences would just get longer and longer. Clearly any decoder is going to have to make a final choice sometime. Intuitively, it is clear that, if one imposes a final decision on branch i only when the decoder has gotten far enough beyond branch i, the probability of making an otherwise unnecessary error at this point can be made negligible. Rigorously, let us suppose that the decoder chooses the single most likely sequence out to branch i and fixes on the first letter of that sequence as its final choice for the first letter. Let us suppose further that no decoding error on the first branch would be made by a maximum likelihood decoder; that is, all sequences which unmerge at the first branch and subsequently remerge have less likelihood than the correct sequence. Then the probability of error is the probability that some sequence which

differs from the correct sequence in its first letter and does not subsequently remerge before branch i has greater likelihood than the sequence of greatest likelihood which agrees with the correct sequence in the first letter; this probability is clearly less than the probability that some one of the totally unmerged sequences is more likely than the correct sequence itself. But there are approximately

$$|S_{1i}| \doteq q^i \tag{101}$$

such totally unmerged sequences, forming a random block code (with condition 2') of length N = ib and rate R = r, so that

$$\begin{aligned} Pr(\mathcal{E}) &\doteq \exp - NE(R) \\ &= \exp - ibE(r) \end{aligned} \tag{102}$$

Let us then choose i so that

$$\frac{i}{\nu} = \frac{e(r)}{E(r)}; \tag{103}$$

that is, to be the same multiple of a constraint length as the tree code exponent is of the block code exponent at rate r; then (102) becomes

$$Pr(\mathcal{E}) \doteq \exp - \nu be(r) \tag{104}$$

and the error probability is not increased by imposing a final decision at branch i. We comment that this shows that the advantage in performance of tree codes over block codes cannot be achieved without this sort of suspended judgment, and that one may expect that tree code decoding algorithms which make final decisions after a constraint length will have no better performance than block codes of the same length.

As for resynchronization, we propose the following modification of Viterbi's algorithm. Pick an initial branch at which to start. There

will be $q^{\nu+1}$ possibilities for the source letters corresponding to that branch and the ν preceding. Among the q of these which have the same final ν letters, choose that which has greatest likelihood on the first branch alone; these choices for each of the q^{ν} possible terminations give q^{ν} survivors. Continue normally with the Viterbi algorithm, calculating likelihoods only from the initial branch onward. This proposed resynchronizing Viterbi algorithm differs from the original only in starting the computation of likelihoods at an arbitrary branch. In terms of the trellis picture, this algorithm amounts to gluing all q^{ν} starting nodes together as an initial node.

We define the algorithm to be resynchronized when all q^{ν} survivors have remerged with the correct path at some point. Suppose we assist the resynchronization by taking the single most likely of the q^{ν} survivors at branch i and discarding all survivors which do not merge with it somewhere between the initial branch and branch i. The probability that the correct sequence is discarded during this maneuver is the probability that after i branches some sequence which is nowhere merged with the correct sequence is the most probable, but as we have seen, this probability is

$$\Pr(\mathcal{E}) \doteq \exp - ibE(r) \tag{105}$$

Thus if i is large enough, the probability of not getting resynchronized by branch i can be made as small as one likes. Of course this maneuver would not be necessary in practice; totally unmerged sequences would be carried along as excess baggage until they either merged or became totally improbable and were discarded.

Note that resynchronization is therefore achievable without any additional decoder complexity over the q^{ν} normally required. It is true that if the decoder computational load is not fixed at q^{ν}, as with the Viterbi algorithm, but variable, as with the sequential Viterbi algorithm or the Fano sequential decoding algorithm, the computation will tend to start off at its maximum value of q^{ν} and then decrease to its normal statistical behavior as resynchronization is achieved. However, if the

maximum computational load L and the constraint length ν have been chosen so that L is somewhat greater than q^{ν}, as recommended in the previous section, then there seems no reason why the decoder should not get over this initial transient and resume normal decoding.

We conclude that an unterminated tree code can be decoded without any loss in performance by a decoder which makes final decisions some fixed number of constraint lengths after it first gets to a branch, and which resynchronizes by simply starting decoding, counting likelihoods from an arbitrary initial branch, as long as the maximum absorbable decoding load is somewhat greater than q^{ν}, so that decoding errors predominate over buffer overflows.

Summary and Conclusions

Let us briefly review our more important results. We have determined the exact exponential behavior of the probability of error of random tree codes with maximum likelihood decoding at all rates. We have showed that, when considered as block codes, terminated random tree codes can be made to give precisely the same performance as random block codes at all rates, this performance being the optimum performance at high rates; then we have exhibited a tree code decoding algorithm less complex than the equivalent block code maximum likelihood algorithm, namely the Viterbi algorithm, thereby showing just wherein the superiority of tree codes over block codes lies. We have observed that performance superior to that of block codes can be obtained if the tree code is not terminated, and have demonstrated that the Viterbi algorithm can be successfully modified to obtain resynchronization and to use only finite memory when the tree code is unterminated.

By showing that buffer overflow and decoding error are asymptotically identical events when the maximum computational load $L \doteq q^{\nu}$ and the rate is at least R_{comp}, we have elucidated the basic interrelationship between these two events, and drawn the moral that in sequential decoders L and ν ought to be chosen so that this equation approximately holds. We have estimated the computational distribution with finite constraint length codes and concluded that computational loads much greater than q^{ν}

become very unlikely, so that buffer overflow may be made rare. The sequential decoding system suggested by these results would therefore be one of only moderate constraint length, with undetected errors predominating over overflows, and with no termination, the rare overflow and initial startup being handled by an automatic resynchronizer in which the decoder would just simply start decoding under each possible assumption about previous information bits.

The most obvious gap in these arguments is the failure to establish results for Viterbi-like algorithms for rates below R_{comp} identical to those known to hold for sequential decoding, in particular to show that the computational distribution can be made to be $L^{-\rho(r)}$ for all rates. As we have conjectured earlier, the path to this result probably consists of setting up a list-of-L Viterbi algorithm, and obtaining expressions for the probability of tree code list decoding error which would relate to the block code list results by the construction of Figure 3, but we have been stymied by the necessity of accounting for the possibility of the L incorrect sequences merging among themselves. This certainly appears to be one of the most interesting open questions.

REFERENCES

Gallager, R. G., "A Simple Derivation of the Coding Theorem and Some Applications," IEEE Trans. Info. Thy., IT-11, 3-18 (1965).

Jacobs, I. M., and E. R. Berlekamp, "A Lower Bound to the Distribution of Computation for Sequential Decoding," IEEE Trans. Info. Thy., IT-13, 167-174 (1967).

Jordan, K. L., Jr., "The Performance of Sequential Decoding in Conjunction with Efficient Modulation," IEEE Trans. Comm. Tech., COM-14, 283-297 (1966).

Shannon, C. E., R. G. Gallager, and E. R. Berlekamp, "Lower Bounds to Error Probability for Coding on Discrete Memoryless Channels," Info. Control, 10, 65-103 (1967).

Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. Info. Thy., IT-13, 260-269 (1967).

Wozencraft, J. M., and I. M. Jacobs, Principles of Communication Engineering, John Wiley & Sons, New York, 1965; Chapter 6.

APPENDIX B. HORSEBACK ANALYSIS OF CONCATENATION SCHEMES

In this appendix we develop a point of view about concatenated sequential decoding schemes, and use it in the invention and rough analysis of a number of such schemes. We are guided by the considerations of Appendix A.

Development of a Point of View

In Appendix A we saw that with a tree code of large constraint length the essential limitation of sequential decoding schemes is the statistical behavior of the decoding computational load. In fact, the number of decoding computations per branch C follows the Pareto distribution:

$$C \doteq K L^{-\alpha(r)}, \tag{1}$$

where K is some constant, (experimentally, of the order of magnitude of 1), and $\alpha(r)$ is the Pareto exponent, which depends explicitly on the tree code rate r and implicitly on the channel statistics through Gallager's function $E_0(\rho)$. We showed in Appendix A that for $r \geq R_1 = R_{comp}$, $\alpha(r)$ is the solution to

$$r = \frac{E_0(\alpha)}{\alpha} = R_\alpha ; \tag{2}$$

this relation apparently holds at all rates for sequential decoding. A variable with a Pareto distribution in which the exponent α is less than 1 has an unbounded mean; a finite constraint length does bound the mean, as we saw in Appendix A, but it is still true that for $\alpha \geq 1$ the mean is relatively small while for $\alpha < 1$ it becomes very large. In fact, the rate r at which $\alpha = 1$, namely $R_{comp} = R_1 = E_0$ (1), seems to represent an upper limit to the practically usable rate of a sequential decoding scheme. This rate is always less than capacity and sometimes significantly so. For example, on a white gaussian channel, in the limit of arbitrarily great bandwidth and

arbitrarily fine receiver quantization, the channel is characterized by the $E_0(\rho)$ function for a very noisy channel:

$$E_0(\rho) = \frac{\rho}{1+\rho} C \tag{3}$$

where C is the rate that would achieve a signal-to-noise ratio per information bit E_b/N_0 of $\ln 2 = .69 = -1.6$db, the Shannon limit for a white gaussian channel. Substituting $\rho = 1$, we find that the sequential decoding limit is $E_0(1) = C/2$; thus sequential decoding seems to be practically bounded 3 db away from channel capacity.

If we wish to get close to capacity, and in deep space telemetry we want every decibel we can get, we must find a way of taming the computational distribution so that it does not blow up at rates near capacity. Let us therefore recall the cause of the Pareto behavior of the decoding computation. We saw in Appendix A that a decoding load of q^ν was to be expected whenever a decoder of decoding constraint length ν would have made an error; further, that with constraint length ν most error events involved unmerged sequences of length ν/μ_0 (in branches), where

$$\mu_0 = 1 - \frac{\alpha E_0'(\alpha)}{E_0(\alpha)} = 1 - \frac{R(\alpha)}{R_\alpha}, \tag{4}$$

α being the solution to (2), and that such error events had probability

$$P_\nu(\varepsilon) \doteq \exp -\nu b e(r) = \exp -\nu b E_0(\alpha). \tag{5}$$

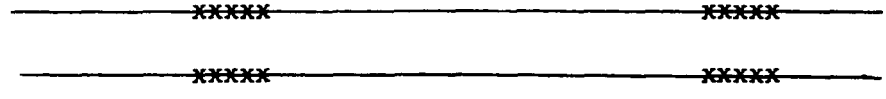
Thus, as had been pointed out by earlier authors, we conclude that

1. The statistics are predominantly due to the channel maintaining a certain critical noisiness for a time τ;
2. The probability of a channel burst of the critical noisiness and length τ goes down exponentially with τ, while the resulting computational load goes up exponentially with τ, the ratio of these exponents being the Pareto exponent.

Suppose we then have a sequential decoder whose speed advantage is significantly greater than the average computational load, and whose maximum computational capacity (speed advantage times buffer size) is L. The above conclusions lead us to believe that all computational failures are due to a simple type of channel misbehavior, namely the channel maintaining the critical noisiness for the critical length of time required to cause just greater than L computations. Other types of channel noise are disregarded for the following reasons:

1. Noise bursts of the critical density but shorter than the critical length cause fewer than L computations; the decoder buffer will fill partially, but since the speed advantage is greater than the mean load, any such transients will most likely be dissipated by the time the next burst arrives.
2. Noise bursts of the critical density but longer than the critical length are very rare, since the probability of such a noise burst goes down exponentially with length.
3. Noise bursts of other than the critical density which lead to fewer than L computations can be disregarded for the reasons in 1.
4. Noise bursts of other than the critical density which lead to L or more computations are rare, because they are governed by an exponential distribution with a greater exponent [$e(r,\mu)$] than the exponent $e(r,\mu_0)$ which applies to the critical density.

There results a simplistic picture which is useful for visualizing channel behavior. Representing a rate - 1/2 code as two parallel streams, for example, we have the picture below:



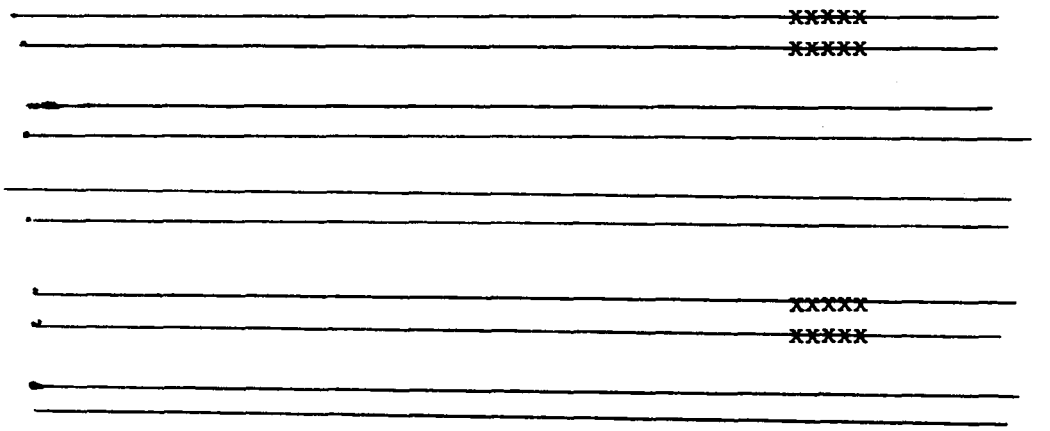
The cross-hatched sections represent noise bursts of just the right critical length and density to cause a computational failure; all such events will be identical, and no other events need be considered.

Example of Horseback Analysis

To show how the above point of view is useful in estimating the performance of concatenation-type schemes, we apply it to the first scheme of this class to be proposed and examined closely, the 'hybrid' scheme of Falconer [1967].

Let the incoming information stream be separated into N-1 parallel streams, and create an Nth parallel stream by forming the mod 2 sum (parity check) of all information streams. Encode these N streams separately, for example in rate-1/2 systematic codes. Decode all these streams separately in parallel; however, when N-1 of the bits entering into any one parity check have been decoded (with acceptably high reliability), cease decoding the Nth laggard stream, and simply calculate its value from the N-1 known bits.

With this scheme a computational failure will occur only if it would have occurred in two of the independent parallel streams simultaneously. Thus to a failure belongs the following picture of the noise (with N = 5):



Of course there are $\binom{N}{2}$ different ways in which such a noise burst can occur on 2 of the N streams. Nonetheless, we see that for a computational failure to occur, noise bursts of the critical density but involving twice the number of bits are required. But the probability of such a burst goes down exponentially with the number of bits involved, and is therefore equal to the square of the original probability of decoding failure. Thus if the original probability was proportional to $L^{-\alpha}$, the new probability will be proportional to

$$(L^{-\alpha})^2 = L^{-2\alpha} \tag{6}$$

The effect is therefore to multiply the Pareto exponent by a factor of 2.

This result could also have been obtained by simply noting that the two overflow events are independent, and therefore

$$(L^{-\alpha})^2 \leq \Pr \text{ (overflow)} \leq \binom{N}{2} (L^{-\alpha})^2, \tag{7}$$

where the lower bound comes from looking at two particular streams, and the upper, from the union bound. The previous analysis has the advantage, however, of being usable in cases where several parallel decoding processes are not independent, as will be the case below.

Falconer-Type Schemes

We now introduce a variety of concatenation schemes that have occurred to us, and use the point of view developed above to predict their performances as best we can.

First, the independent channel type of scheme used as an example above is obviously extendable to more elaborate precoding than a simple parity check. Let there be K information streams; from them form N-K parity check streams by use of an (N, K) block code which has minimum distance D. Encode and decode each of the N resulting streams separately, using the block code to pick up any D-1 laggard streams. The resulting probability of decoding failure is proportional to $L^{-D\alpha}$; thus arbitrarily large exponents can be obtained. The overall rate is equal to rK/N, which for fixed D can be made as close as desired to the original rate r by increasing N. This is the general class of schemes considered by Falconer [1967]; we earlier [1966] made estimates of the parameters of such schemes suitable for the white gaussian channel, and determined the performance to be expected.

Implementation of this class of schemes is somewhat messier than might be desired. In order to minimize the rate loss required to obtain a certain D, one would use a non-binary Reed-Solomon code, where K equals N-D+1, the maximum value possible. Although encoding and erasure correction with such a code are straightforward, they are operations of a character quite different from sequential decoding, involving

finite field manipulations. In particular, erasure-correction requires an algorithm involving a sequence of finite field multiplications (proportional to N or to D^2, whichever is larger) and a finite field division. Although finite field operations are particularly well suited to digital hardware, the need to introduce such a totally different decoding procedure may be expected to increase decoding complexity rather significantly. An alternative is to use an easily decoded binary code; binary codes are less efficient, however, and will therefore result in a larger rate loss or larger N.

Variations of the Independent Subchannel Scheme

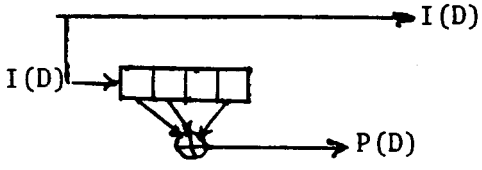
Next, we propose variations on the above class of schemes. It will be helpful here and in the sequel to introduce the delay operator notation for data streams; a series of bits $i_0, i_1, i_2, i_3, \ldots$ is represented by the polynomial

$$I(D) = i_0 + i_1 D + i_2 D^2 + i_3 D^3 + \cdots \tag{8}$$

in the delay operator D. If we have N-1 information streams, for example, we can denote them $I_1(D)$, $I_2(D)$, ..., $I_{n-1}(D)$. An overall parity check on these streams is simply

$$S(D) = \sum_{i=1}^{N-1} I_i(D), \tag{9}$$

where the sum is modulo 2. A convolutional encoder with input I(D) and shift register taps represented by G(D) produces a parity check P(D) = I(D)G(D), where again all operations are modulo 2; for example, the encoder



has the generator polynomial $G(D) = 1+D+D^3$, and P(D) is related to I(D) by

$$P(D) = (1+D+D^3)I(D). \tag{10}$$

In the scheme treated in the example, the N-1 information streams $I_i(D)$ and the sum stream S(D) were each encoded with a rate-1/2 systematic convolutional encoder, so that 2N streams in all were transmitted. If we suppose that all generators are the same, say G(D),

then the parity streams generated are

$$P_i(D) = G(D)I_i(D), \quad 1 \le i \le N-1;$$

$$P_s(D) = G(D)S(D). \tag{11}$$

Correspondingly, let the received streams be denoted by $I_i'(D)$, $S'(D)$, $P_i'(D)$, and $P_s'(D)$, and the error sequences by

$$E_{Ii}(D) = I_i'(D) + I_i(D)$$

$$E_s(D) = S'(D) + S(D)$$

$$E_{pi}(D) = P_i'(D) + P_i(D)$$

$$E_{ps}(D) = P_s'(D) + P_s(D). \tag{12}$$

Continuing with this same example, consider the time when all but two of the parallel pairs have been decoded up to some point, say $I_1'(D) - P_1'(D)$ and $I_2'(D) - P_2'(D)$. [Any two would do as well, including $S'(D) - P_s'(D)$]. Presumably we know the correct values of $S(D)$ and $I_i(D)$, $i \neq 1, 2$, and hence also

$$S(D) + \sum_{i=3}^{N-1} I_i(D) = I_1(D) + I_2(D). \tag{13}$$

Let us then form the two streams

$$I_2''(D) = I_2'(D) + I_1(D) + I_2(D)$$

$$P_2''(D) = P_2'(D) + [I_1(D) + I_2(D)]G(D). \tag{14}$$

By simple substitution

$$I_1'(D) = I_1(D) + E_{I1}(D)$$

$$P_1'(D) = I_1(D)G(D) + E_{p1}(D)$$

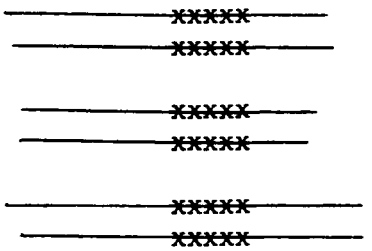
$$I_2''(D) = I_1(D) + E_{I2}(D)$$

$$P_2''(D) = I_1(D)\,G(D) + E_{p2}(D). \tag{15}$$

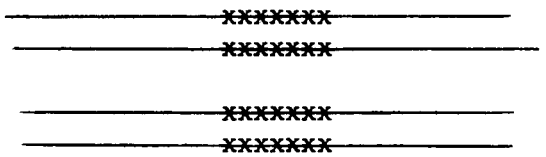
These four streams may therefore be decoded together as a rate-1/4 code for $I_1(D)$; $I_2(D)$ can of course then be determined from the parity check.

What sort of exponent does this variation have, in comparison to the 2α which the original scheme would give? There are now two

types of computational failures. The first occurs when there would have been a computational failure in three or more streams independently:



and thus has an exponent 3α . The second occurs when two streams have a noise burst so bad that it defeats a rate-1/4 code:



which event will have the exponent $\alpha(1/4)$ appropriate to a rate-1/4 code. Thus the question is whether $\alpha(1/4)$ exceeds $2\,\alpha(1/2)$. Now $\alpha(R)$ is the solution to

$$R = \frac{E_0[\alpha(R)]}{\alpha(R)}; \tag{16}$$

thus

$$\alpha(1/4) = 4E_0[\alpha(1/4)] \geq 4E_0[\alpha(1/2)] = 2\alpha(1/2), \tag{17}$$

where we use the fact that $E_0(\rho)$ is a non-decreasing function of ρ. Thus, we always improve the exponent with this particular stratagem. No additional rate loss is involved; there is however, a need for a separate rate-1/4 decoder, involving additional decoding complexity.

One could further consider intervening in this way when N streams remained undecoded, N . 2. The resulting exponent would then be

$\min \left\{ (N+1)\,\alpha(1/2),\ \alpha[1/2(1-1/N)] \right\}$ for a rate-1/2 code, or

$$\min \left\{ (N+1)\,\alpha(R),\ \alpha[R(1-1/N)] \right\} \tag{18}$$

in general for a code of rate R. Since the former quantity increases

with N while the latter decreases, increasing N pays as long as

$$\alpha[R(1-1/N)] > N\alpha(R), \tag{19}$$

or

$$\frac{E_o\{\alpha[R(1-1/N)]\}}{R(1-1/N)} > \frac{NE_o[\alpha(R)]}{R};$$

$$E_o\{\alpha[R(1-1/N)]\} > (N-1)E_o[\alpha(R)]. \tag{20}$$

This will always hold for N=2, as above, but higher values of N are justified only at the rates closest to capacity.

Finally, the same stratagem can be used when there is more than one redundant stream; the analysis is similar.

A final variation would be to precode with a convolutional code of rate K/N capable of correcting D-1 erasure bursts, rather than with a block code. This would also give an exponent of $D\alpha$, and might very well have some implementational advantages. Whether or not it does so depends on the properties of such codes, which we have not had time to examine.

Mixed Subchannel Schemes

By not transmitting the redundant information stream, we can cut the rate loss at the cost of a lesser exponent. An example of such a scheme is the following. Let the information streams be $I_i(D)$, $1 \le i \le N-1$. Let

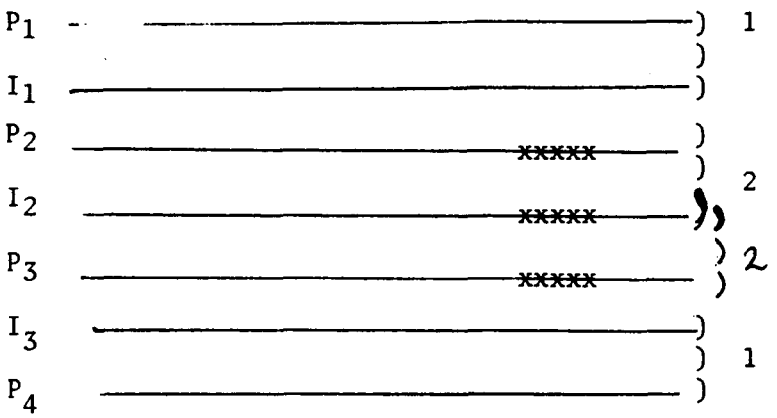
$$\begin{aligned} P_1(D) &= I_1(D)G(D) \\ P_2(D) &= [I_1(D) + I_2(D)]G(D) \\ P_i(D) &= [I_{i-1}(D) + I_i(D)]G(D), \quad 2 \le i \le N-1 \\ P_N(D) &= I_{N-1}(D)G(D), \end{aligned} \tag{21}$$

where G(D) is some generator polynomial of a rate-1/2 systematic convolutional code. The overall rate is now (N-1)/(2N-1), which is closer to 1/2 than the independent subchannel scheme above for the same N. The decoding procedure for such a scheme would be to start by simultaneously decoding $I_1'(D) - P_1'(D)$ and $I_{N-1}'(D) - P_N'(D)$ as rate-1/2

codes; when $I_1(D)$ is decoded, then form

$$P_2''(D) = P_2'(D) + I_1(D)G(D)$$
$$= I_2(D)G(D) + E_{p2}(D) \qquad (22)$$

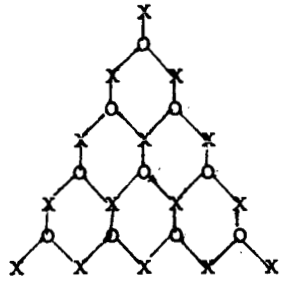
and decode $I_2(D)$-$P_2''(D)$, and similarly work up from the bottom; stop when all I (D) have been decoded from one direction or the other. A typical failure pattern might then be (for N=4):

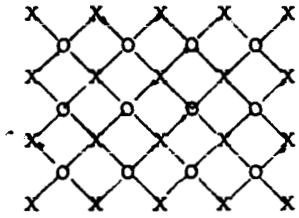


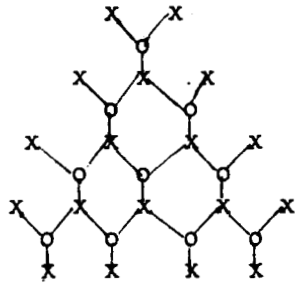
We see that if one received parity stream is particularly noisy it can be ignored. This scheme will be frustrated if both the parity streams associated with an information stream and the information stream itself have the critical length and density, as illustrated above; the bits involved are 3/2 the number ordinarily involved, so we would expect an exponent of $(3/2)\alpha$ for this scheme.

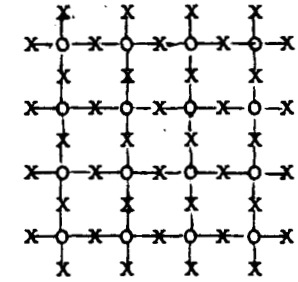
This scheme can be improved in the same way as was done above, by invoking a rate-1/3 decoder when only one information stream and two parity streams remain to be decoded, or in general a rate $N/(2N+1)$ decoder when only $2N+1$ streams remain. As above, the rate-1/3 decoder will always improve the exponent, but earlier interventions are justified only at rates near capacity.

These ideas can be extended to give greater exponents in an interesting but probably unprofitable way. A few examples will show the general principle. Consider the regular geometrical constructions below:









In these figures the symbol o represents an information stream, and x represents a parity stream. The parity streams are formed by taking the mod 2 sum of the information streams connected to the parity stream by the lines of the figures and encoding the sum with a convolutional encoder whose generator polynomial is G(D):

$$P(D) = [\sum I_i(D)]G(D). \qquad (23)$$

Decoding is begun at the corners in the upper figures and at the borders in the lower figures, and continues by the 'subtracting out' of already decoded streams, as above. Each one of the information streams can be approached from d different directions, where $d = 3$ on the left and $d = 4$ on the right; it is fairly easy to convince oneself that no computational failure can occur unless there are noise bursts of the critical length and density in at least d parity streams and one information stream, so that the expected exponent would be $[(d+1)/2]\alpha$. However, the excess of the number of parity streams over information streams, rather than being constant, goes up as the square root of the number of information streams. Therefore, while the overall rate will approach 1/2 with increasing numbers of streams, it does so relatively slowly, so that one would expect relatively large patterns, implying decoding complexity.

These same patterns can be used in an independent subchannel type of scheme by generating more than one parity stream for each x and not transmitting the information stream. In this case the resulting exponent will be $(d+1)\alpha$; it is apparent that this is just a geometric way of constructing a code of minimum distance $D=d+1$, and not a very efficient code at that, although with it erasure-correction is very simple.

Reverse Decoding

Ordinarily convolutional codes are periodically resynchronized by the insertion of a constraint length of fixed dummy bits in the information stream. Convolutional codes may also be automatically resynchronized, as we saw in Appendix A. With either of these methods, the possibility exists of starting at a point at which the decoder is resynchronized and decoding backwards. With a systematic code, such reverse decoding can be expected to have some of the effects of concatenation, but without any rate loss in comparison to the conventional system.

Consider a rate-1/2 systematic code of constraint length ν; that is, the encoding polynomial G(D) has degree ν. Writing out the encoding equation, we have the convolution

$$p_k = \sum_{j=0}^{\nu} g_j i_{k-j}, \tag{24}$$

where g_0 and g_ν are always equal to 1. Normally, a sequential decoder hypothesizes the information bits in order of increasing subscript; each new hypothesis then depends on examination of the received bits

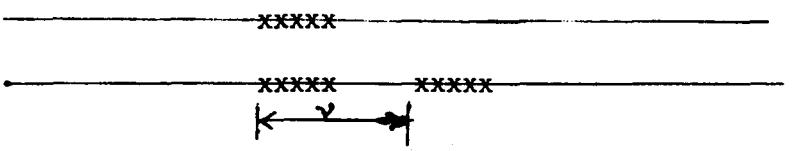
$$\begin{aligned} i'_k &= i_k + e_{ik} \\ p'_k &= i_k + \left(\sum_{j=1}^{\nu} g_j i_{k-j}\right) + e_{pk}, \end{aligned} \tag{25}$$

if the quantity in parentheses is assumed known, then we have two separate estimates of i_k, which permit a reasonable hypothesis. On the other hand, when decoding in the reverse direction, the information bits are hypothesized in order of decreasing subscript, so that the

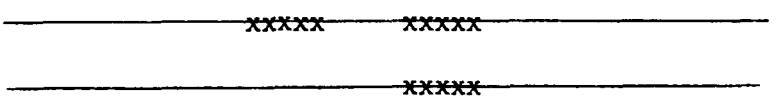
hypothesis of i_k must be based on the received bits

$$i'_k = i_k + e_{ik}$$
$$p'_{k+\nu} = i_k + \left(\sum_{j=0}^{\nu-1} g_j i_{k+\nu-j}\right) + e_{p,k+\nu}, \qquad (26)$$

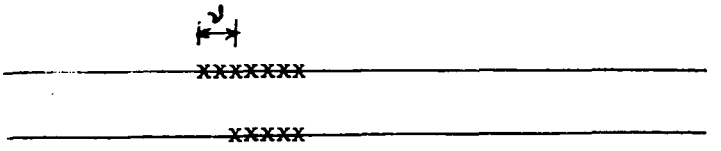
Where again the quantity in parentheses may be assumed known at the time of hypothesizing i_k. We observe that the parity bit associated with i_k is skewed by ν bits in the reverse decoding; if ν is large enough, this suggests that the forward and backward parity streams associated with the information stream may be almost independent. Suppose then we adopt the strategy of decoding from both ends simultaneously. In the terms of our horseback analysis, the patterns required for computational failure would be



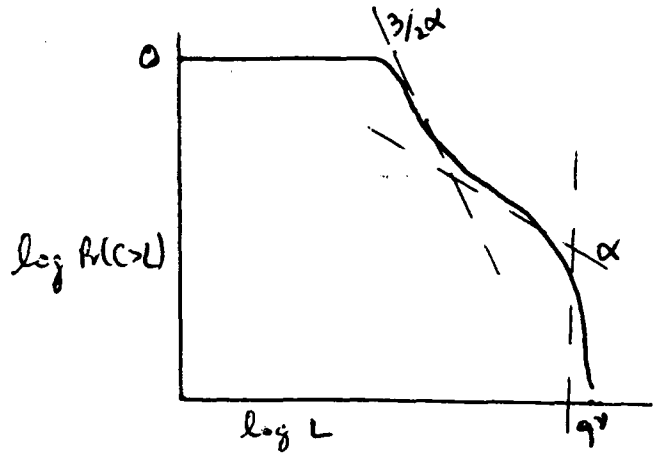
or



which would lead to an estimate of $(3/2)\alpha$ for the Pareto exponent, for small values of the computational variable L. For larger values of L, the critical length will begin to exceed the constraint length ν, as we saw in Appendix A; then the failure pattern will be



and the exponent should begin to decrease toward α. Finally, for $L > q^{\nu}$, the computational distribution will begin to fall off very rapidly, as it does in the ordinary case. Thus the general shape of the curve of $\log \Pr(C>L)$ versus $\log L$ would be expected to be



The hope would be that the constraint length would be sufficiently long that the range of L in which α is multiplied by 3/2 would be the range of practical interest.

One virtue of reverse decoding is that it may be applied in combination with any of the other concatenation techniques without much additional complexity or any additional rate loss, to give an effective multiplication of the exponent otherwise attainable by 1.5. Thus although the expected gain is not remarkable, it comes almost for free, and can be obtained in addition to whatever can be obtained otherwise; for example, it would seem almost certainly preferable to get an exponent of 3α by using one of the earlier schemes to get 2α and using reverse decoding than by relying on one of the earlier schemes by itself.

We have been able to extend the reverse decoding idea with a geometrical approach like that described earlier, which is again interesting though not necessarily useful. Imagine an encoder which first lays out the information bits in a n-dimensional regular array, such as the 2-dimensional rectangular array below.

. . . . . .
. . . . . .
. . o o o o o . . o o o o o . . o
. . o o o o o . . o o o o o . . o
. . o o o o o . . o o o o o . . o
. . o o o.o o . . o o o o o . . o
. . . . . .
. . . . . .
. . o o o o o . . o o o o o . . o

Here the open circles are information bits, and the dots fixed dummy bits; the latter form borders around blocks of information bits. Now let there be one parity bit for each information or dummy bit in this infinite array, and let each parity bit be the mod 2 sum of a fixed pattern of bits in the corresponding square three bits on a side. For example, if we label the information bits i_{jk}, then we might have

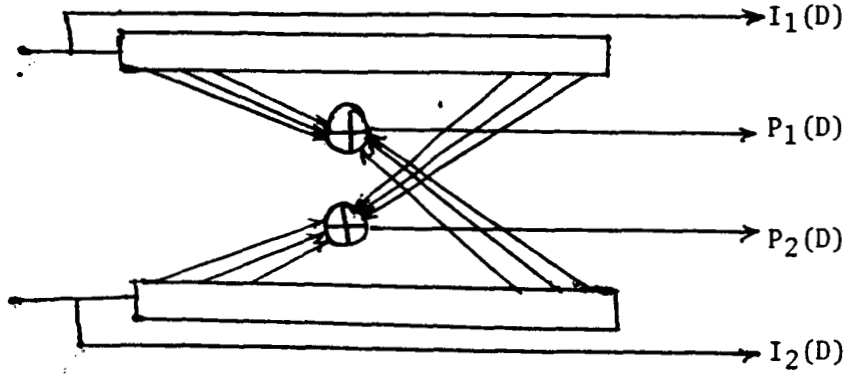
$$P_{jk} = i_{jk} + i_{j,k+2} + i_{j+1,\,k+1} + i_{j+2,k} + i_{j+2,k+2}, \qquad (27)$$

making sure that all the corners of the square are represented. Then it is possible to begin decoding any block at any corner, and any bit may be reached in decoding from any of 4 different directions. In each direction a different parity bit is associated with the hypothesis of a particular information bit. Although it is not obvious how best to construct a sequential decoder for such a code, other than setting an ordinary decoder to run along rows and columns, we might hope for an exponent of $(5/2)\alpha$.

This scheme can clearly be extended to higher dimensions and other types of regular arrays. Its major theoretical drawback is the large ratio of border bits to information bits, particularly in the higher dimensions, with a consequently high rate loss.

Cross-Coupled Coding

Finally, we mention a provocative idea which seems to promise some of the virtues of concatenation with no rate loss whatsoever, by using a code which is decodable by two different convolutional decoders. Consider the rate-2/4 code made up of two parallel rate-1/2 codes which is illustrated below:



The first parity stream is the sum of a set of recently transmitted bits from the first information stream, and a set of bits transmitted some time previously in the second information stream; the second parity stream, vice versa. Thus the encoding equations might be

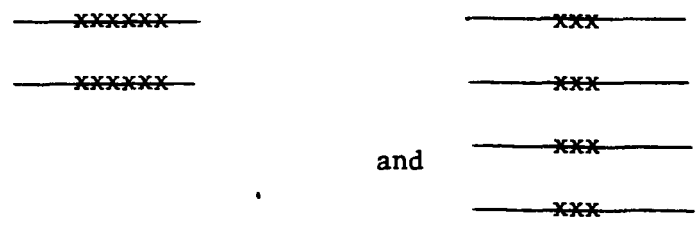
$$P_1(D) = G(D)I_1(D) + G(D)I_2(D)D^n$$
$$P_2(D) = G(D)I_1(D)D^n + G(D)I_2(D). \qquad (28)$$

Decoding would begin with two independent rate-1/2 sequential decoders working on the pairs $I_1'(D)$-$P_1'(D)$ and $I_2'(D)$-$P_2'(D)$. As long as the two decoders advanced at roughly the same rate, the effects of $I_2(D)$ in $P_1(D)$ or of $I_1(D)$ in $P_2(D)$ could be subtracted out. However, if one decoder ran into a difficult search, the other, say the first, would eventually get n bits ahead. Then the strategy would be to switch to a rate-2/4 decoder decoding the four streams

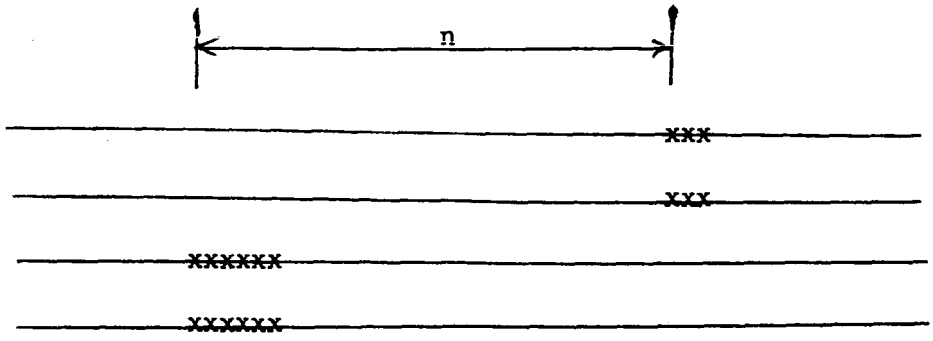
$$I_1'(D) = I_1(D) + E_{I1}(D)$$
$$P_1'(D) = I_1(D)G(D) + G(D)I_2(D)D^n + E_{p1}(D)$$
$$P_2'(D)D^n = (G(D)I_1(D)D^{2n}) + G(D)I_2(D)D^n + E_{p2}(D)D^n$$
$$I_2'(D)D^n = I_2(D)D^n + E_{I2}(D)D^n \qquad (29)$$

for the two information streams $I_1(D)$ and $I_2(D)D^n$. [The $I_1(D)$ information bits 2n bits earlier, in parentheses, would be assumed known.] As soon as $I_2(D)$ was over the hump, the decoder would return to the former strategy.

Estimating the performance of this strategy is a bit tricky. Computational failures with a rate-2/4 code will be caused by noise bursts of the same density but half the length (though the same length in total bits) as with a rate-1/2 code. That is, the two failures look like

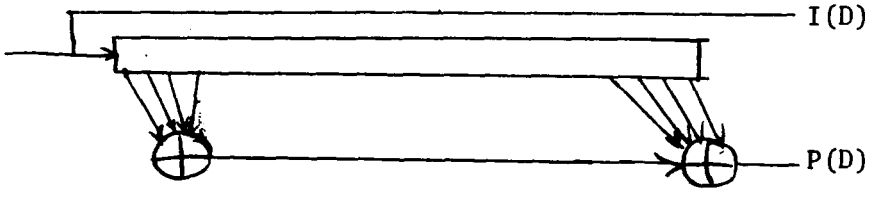


Therefore the cross-coupled code will certainly be defeated by a noise burst of the following form:



Thus the most we can get is an exponent of $3/2\alpha$. However, we may not even get this much. First, the cross-coupled code is a particularly lousy rate-2/4 code, and although the computational behavior of a sequential decoder seems almost totally insensitive to quality, some degradation in the exponent might occur. Second, given a critical noise burst in $I_2'(D)$-$P_2'(D)$, there might be some noise burst in the other pair more likely than that illustrated above which would lead to computational failure.

An alternate application of these ideas, which is somewhat more elegant, but which depends on the ability of a sequential decoder to resynchronize, would involve a simple rate-1/2 code with the following encoder:



In other words,

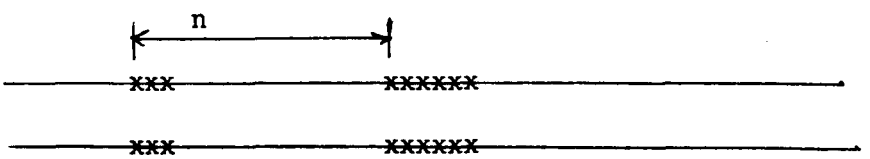
$$P(D) = I(D)G_1(D) + I(D)G_2(D)D^n, \qquad (30)$$

where n is large enough that an ordinary sequential decoder can certainly resynchronize itself in n bits. For instance, a fixed sequence might be inserted every n bits. Ordinarily, the decoder would be

assumed to know the information bits from n time units previous, and would decode $I'(D) - P'(D)$ using only the front end of the shift register; thus as long as no failures occurred the behavior would be precisely that of an ordinary rate-1/2 sequential decoder with the code generated by $G_1(D)$. In the event of computational failure, the decoder would skip ahead and obtain resynchronization, presumably before the undecoded bits reached the right end of the shift register. When they did so, the decoder would consider the four following streams simultaneously:

$$I'(D) = I(D) + E_I(D)$$
$$P'(D) = I(D)G_1(D) + I(D)G_2(D)D^n$$
$$I'(D)D^n = I(D)D^n + E_I(D)D^n$$
$$P'(D)D^n = I(D)G_1(D)D^n + [I(D)G_2(D)D^{2n}] + E_p(D)D^n$$

where the quantity in brackets would be assumed known and subtracted out. These four streams are then the equivalent of a rate-2/4 code in the two information streams $I(D)$ and $I(D)D^n$, and can be decoded as such. As soon as the undecoded section of $I(D)D^n$ is decoded, ordinary rate-1/2 decoding would be resumed. The analysis of computational failures is exactly as above; in particular, the pattern



must lead to failure, and there may be worse, so the improvement is no better than $3/2\alpha$.

Extensions of this approach come to mind immediately; clearly one could add more and more clumps of generators. How to choose the spacing of the generators and the rules for changing modes then becomes quite a complex problem.

In conclusion, the improvement in exponent with these strategies is no more than 3/2, and may not even be that much. However, the ideas involved here are very interesting, in that they show that no rate loss is necessary in principle to realize an improvement in computational behavior; the essential principle seems to be to set up the code so that the decoder can operate in any of a number of alternate modes, thus forcing the noise to gang up on you in several places simultaneously.

Conclusions

In this appendix we have developed a point of view about what causes computational failures in sequential decoding schemes, and we have used this point of view to estimate the performance of a grab bag of different schemes. Most of these schemes, with the exception of reverse decoding and cross-coupled coding, involve some rate loss, and all involve increased decoding complexity; however, all also result in an improvement in Pareto exponent which more than compensates for the rate loss, since in every scheme the rate loss may be made to approach zero without changing the improved Pareto exponent. None of these schemes clearly emerges as the most desirable, however; we simply now have a lot of methods to try.

One has the feeling that the ultimate scheme remains to be invented. The central notion uniting all the schemes discussed is the provision of alternate methods of decoding any particular information bit or stream, so that the noise is forced to be bad in several places at once. The way this is done in all these schemes seems more or less brute force and inelegant; one imagines there must be some way the same effect could be achieved easily and naturally. The existence of such a scheme is certainly the most important question left open by this appendix.

References

Codex Corp., Phase II Report on a Study of Coding for Deep Space Telemetry, Contract NAS2-2874, Watertown, Mass., March 28, 1966; Appendix E.

D. D. Falconer, "A Hybrid Sequential and Algebraic Decoding Scheme", Ph.D. thesis, M.I.T. Dept. of Electrical Engineering, February, 1967.

APPENDIX C

SIMULATIONS

In this appendix, we describe simulations whose purpose was to assess the validity of the analysis methods of Appendix B by comparing actual performance with predictions for a few simple schemes. We shall first describe the schemes we chose to simulate and the features of the simulation programs, and then exhibit and discuss the results obtained.

Basic Coding Scheme

All the schemes described in Appendix B start with an ordinary convolutional code and a sequential decoder which when used on a memoryless channel has a Pareto computational distribution with exponent α. To minimize programming effort and maximize simulation speed, we chose the simplest possible such scheme, a rate-1/2 systematic code used on a binary symmetric channel with variable crossover (error) probability p. The Pareto exponent α of such a scheme is given as the solution to

$$\frac{1}{2} = \frac{E_0(\alpha)}{\alpha}, \tag{1}$$

where Gallager's function $E_0(\rho)$ is given in bits by

$$E_0(\rho) = -\log_2 2^{-\rho}\left[p^{\frac{1}{1+\rho}} + (1-p)^{\frac{1}{1+\rho}}\right]^{1+\rho} \tag{2}$$

for a binary symmetric channel. The solution to (1) is depicted graphically in Figure 1.

Though this basic coding scheme is a simple one, we expect that our results are generally valid, since our analyses in Appendix B led us to believe that the overall Pareto exponent with a concatenation scheme would be some simple multiple of the Pareto exponent of the basic scheme, so that the Pareto exponent is the only important parameter of the basic code.

We are more particularly concerned with the db improvements that can be obtained on the white gaussian channel. We would expect to

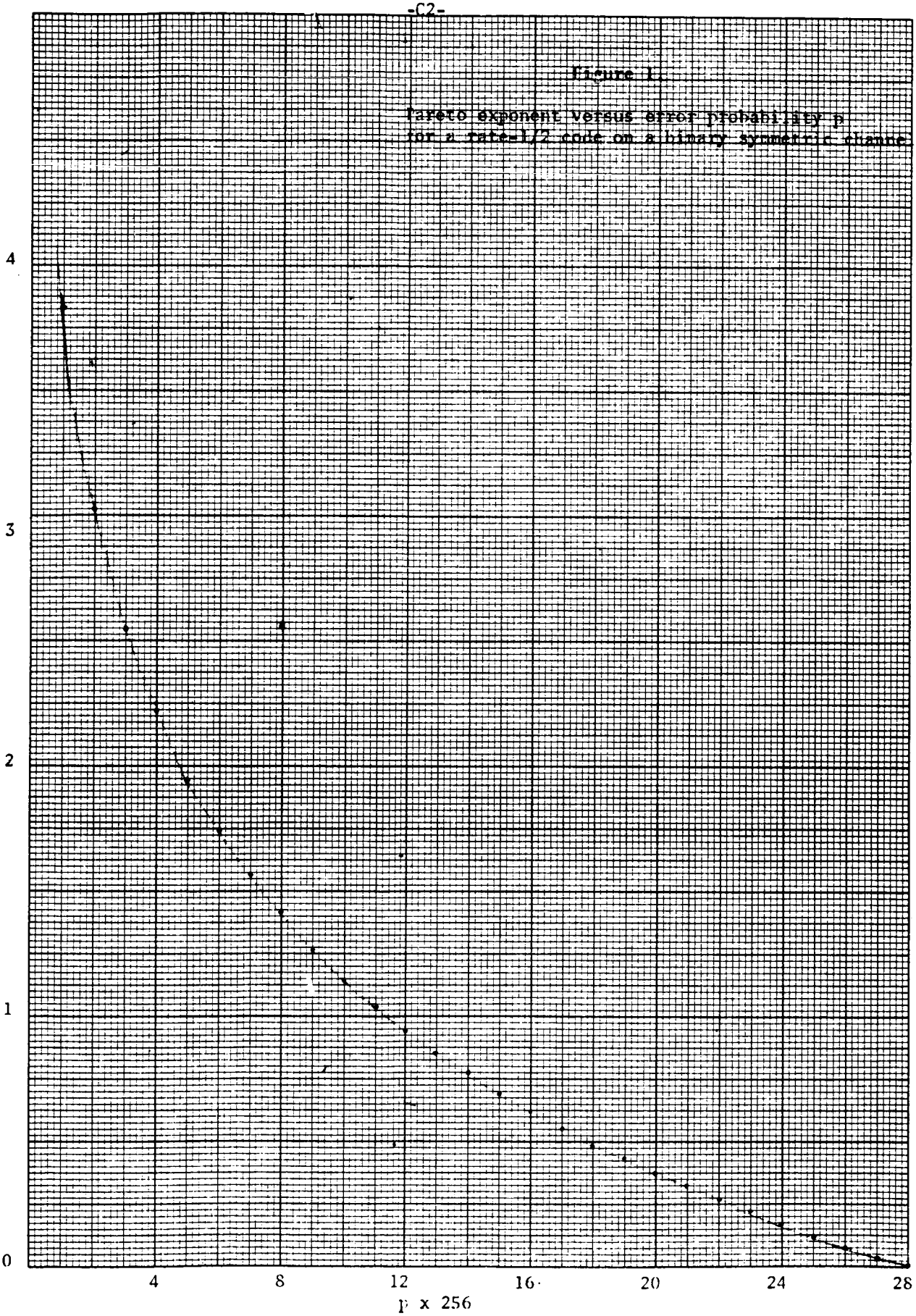


Figure 1.

Pareto exponent versus error probability p
for a rate-1/2 code on a binary symmetric channe

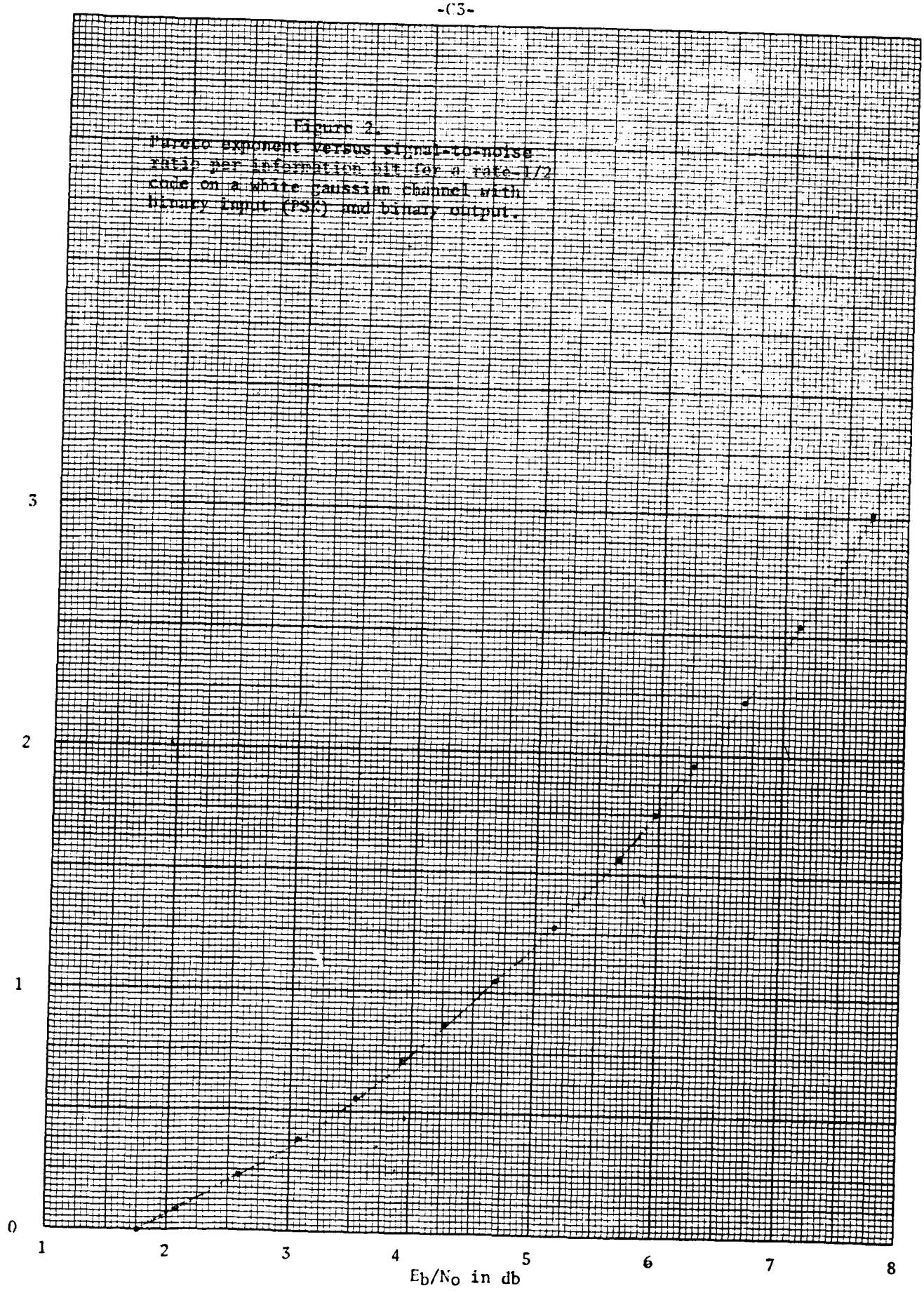


Figure 2.
Pareto exponent versus signal-to-noise ratio per information bit for a rate-1/2 code on a white Gaussian channel with binary input (PSK) and binary output.

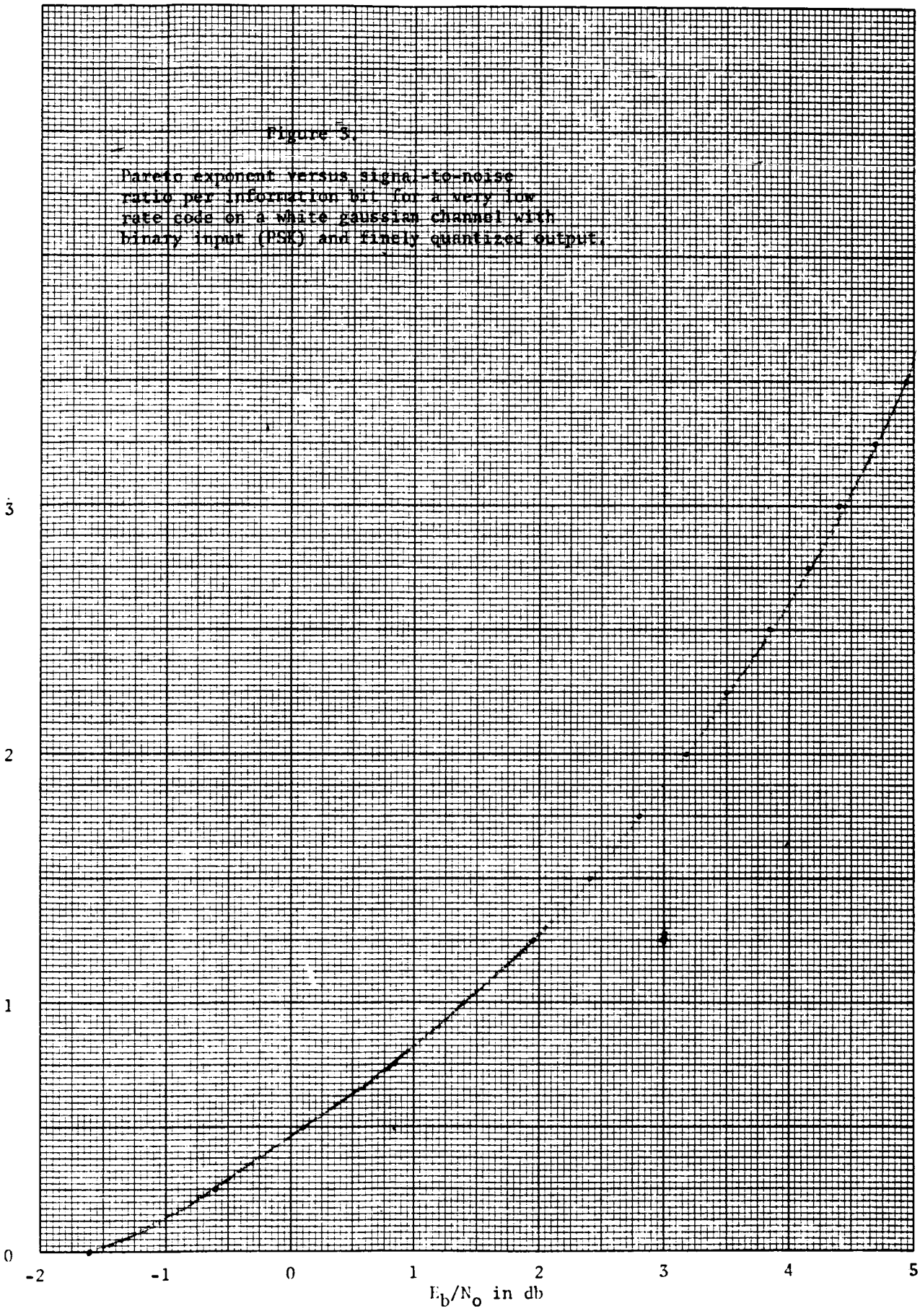


Figure 3.

Pareto exponent versus signal-to-noise ratio per information bit for a very low rate code on a white gaussian channel with binary input (PSK) and finely quantized output.

use a scheme which gave an effective exponent multiplier of M by starting with a basic scheme of exponent 1/M, and getting an over-all exponent of 1; the resulting signal-to-noise ratio per information bit (E_b/N_o) would be that of the basic scheme times the inverse of the scheme's rate loss. Figure 2 gives α as a function of the basic E_b/N_o for a rate-1/2 code with hard decision demodulation on a white gaussian channel. We see that the E_b/N_o at which α =2/3, 1/2, 1/3, and 1/4 are .8 db, 1.2 db, 1.6 db, and 2.0 db respectively below the E_b/N_o at which α =1, which measures the potential gain with schemes of multiplier 3/2 through 4. Of course, normally one would want to use neither rate 1/2 nor hard decision demodulation. However, we believe that these figures will be accurate to within tenths of a db for any PSK scheme. As support for this contention, we sketch in Figure 3 the same curve for the class of schemes in which the rate is very small and receiver quantization very fine, where the results for the very noise channel apply; for this class, which is at the opposite end of the spectrum from the rate-1/2 hard decision scheme, the figures above are accurate to within .1 db.

The basic code used was one of constraint length 33 (ν =32 in the terminology of Appendix A), with the taps of the parity generator given in octal notation by 71547370131. The constraint length was chosen as a natural and convenient length to use with a 16-bit general-purpose computer; the code itself was chosen, from among several of the same length known to be fairly good, as that having the lowest undetected error probability in some short preliminary runs. No consideration was given to its performance when used backwards. Shortened versions of the same code were used to check out hypotheses concerning constraint lengths; these were the length 24 code 7514737 and the (symmetric) length 15 code 71547. Information sequences were arranged into frames of length N up to 512, with the last bits set to a fixed all-zero sequence to simulate resynchronization.

The decoder was a variant of the Gallager version of the Fano algorithm adapted specifically to a rate-1/2 binary output code. The metric increments used were in the ratio of +1 for a bit hypothesis in

agreement with what was received and -9 for a disagreement, or on a per branch basis (with two bits per branch) +2, -8, and -18; this is approximately optimum near p=.045, where α is near one. The threshold spacing was 4, in the same scaling. The ordering of the examination of branches was such that the branch on which the hypothesized bit was equal to the received information bit was always examined first, except that only information bits equal to zero were permitted in the last constraint length. Decoding computations were defined as either forward or backward moves, the former being any memory reference to a branch after the current branch, and the latter the opposite, the definition of current branch changing with each move. Since it follows that, in decoding a frame, forward moves always exceed backwards moves by exactly a frame length, only backwards moves were counted; further, these moves were counted only in units of a computational quantum Q.

Whenever the total number of computational quanta exceeded 2^{16}, an overflow was declared, and decoding on that frame terminated. In all frames in which there was no overflow, the decoded frame was checked for decoding errors, and the total number of information bit errors printed out.

Since the decoding algorithm used a symmetric data-dependent branch ordering, we could assume that the all-zero information sequence was always transmitted without bias to the results. Channel errors were simulated by a random number generator constructed as follows. A primitive polynomial of degree 32, 41760427607 in octal notation, was used as the generator polynomial of a simulated maximum length shift register generating a sequence of period $2^{32}-1$. Successive 8-bit segments of this maximum length sequence were assumed to be independent random integers evenly distributed between 0 and 255. By comparing these integers to a fixed integer, error probabilities ranging from 0 to 255/256 in steps of 1/256 could be obtained.

Schemes Simulated

Three basic schemes were simulated: forward, forward-backward, and side-by-side.

The forward scheme was simply ordinary sequential decoding of the rate - 1/2 systematic code resynchronized every N information bits. This simulation had the purpose of establishing the basic computational distribution and probability of error as a function of code constraint length and channel probability of error.

The forward-backward scheme was a simulation of the simultaneous decoding of a single resynchronized frame from both ends, the reverse decoding of Appendix B. The decoder was time-shared between the two directions of decoding, switching from one to another after each computational quantum of Q backward moves. Decoding terminated whenever the complete frame was decoded in either direction; the computational variable measured was the total number of computational quanta used in the forward direction, or half the total number of quanta.

The side-by-side scheme was a simulation of the simplest mixed subchannel scheme of Appendix B with N=1; in other words, two independent parity streams were generated from a single information stream and all three streams were transmitted. The decoder was time-shared between the decoding of the received information stream and the first received parity stream as one rate-1/2 code, and the received information stream and the second parity stream as another. Again, a switch was made after each computational quantum Q on one pair, and the computational variable measured was the total computational quanta for one of the decodings to terminate, or half the total computational quanta.

Results and Discussion

Figures 4-6 show the computational distribution obtained for each of the three schemes at error probabilities of 9/256, 12/256, 15/256, and 18/256. In all these runs, the frame length was taken to be 128 bits, the code used was the one of constraint length 33, the computational quantum was 4, and the total number of frames was 2^{13}=8192. Thus each run involved about a million (2^{20}) bits; running times on a DDP-116 ranged from 5 to 40 minutes per run. The graphs are log-log; each point represents the total number of frames out of 8192 that the number of computational quanta exceeded 2^n, $n \leq 16$.

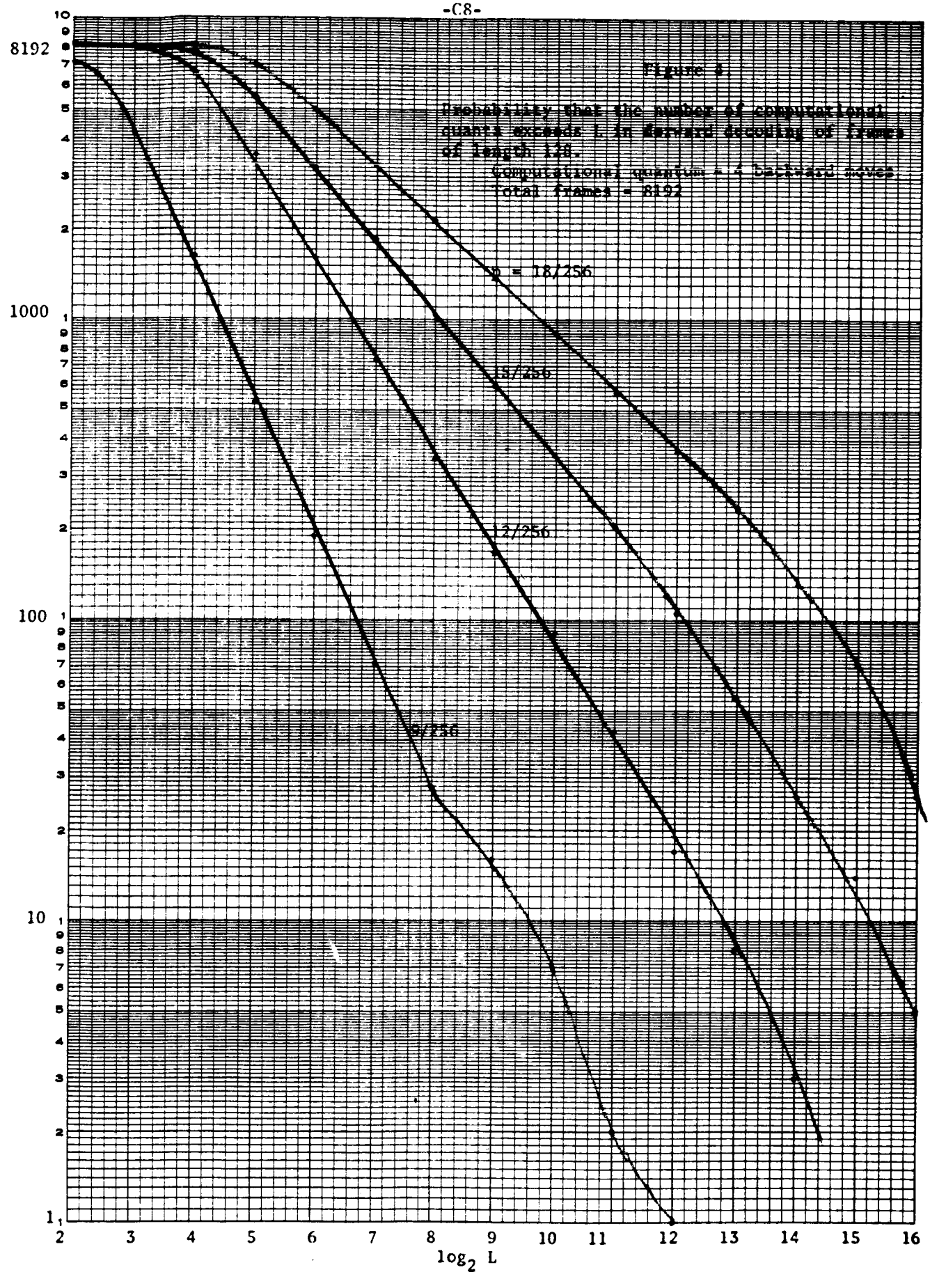


Figure 4.

Probability that the number of computational quanta exceeds L in forward decoding of frames of length 128.

Computational quantum = 4 backward moves
Total frames = 8192

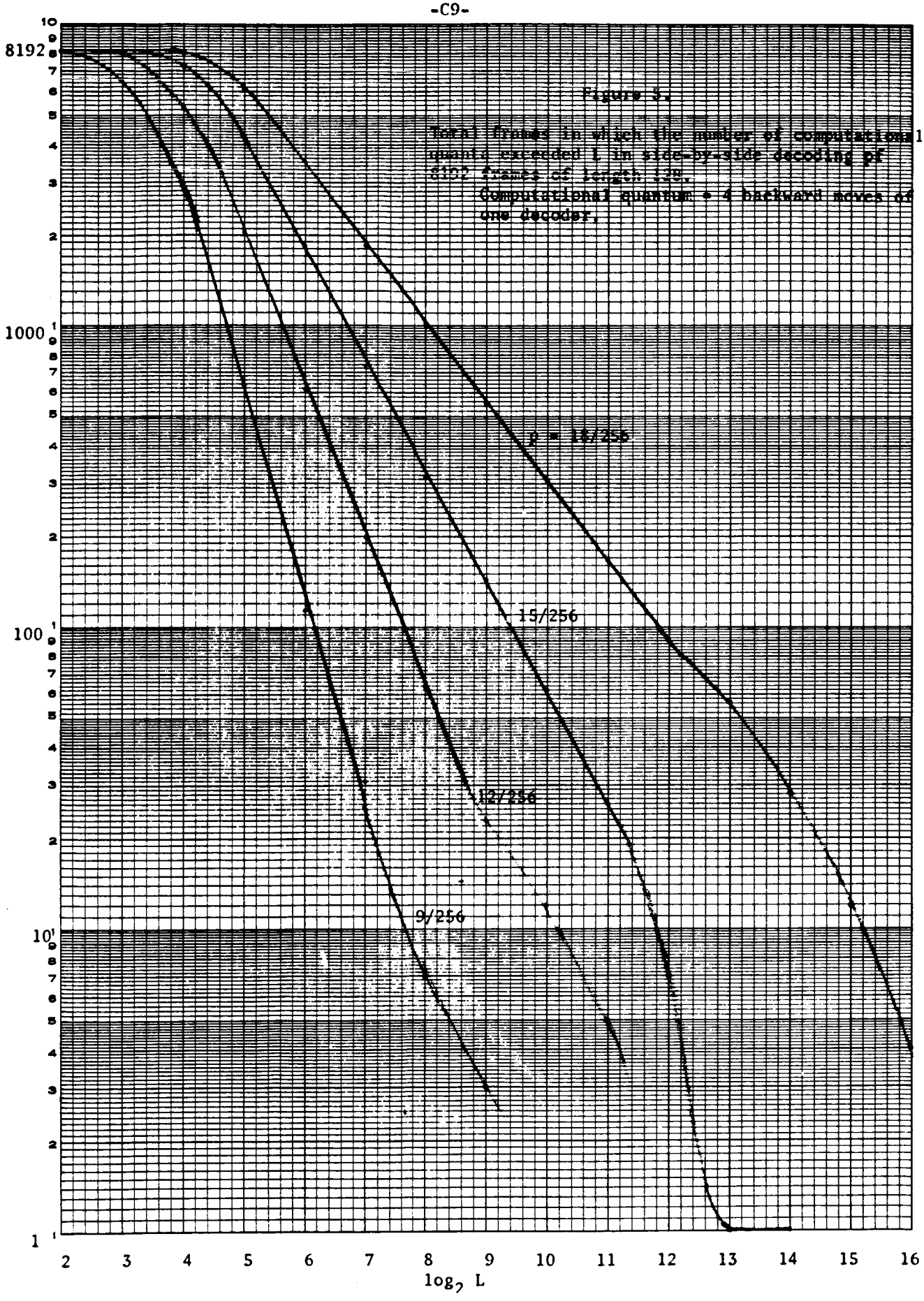


Figure 5.

Total frames in which the number of computational quanta exceeded L in side-by-side decoding of 8192 frames of length 128.
Computational quantum = 4 backward moves of one decoder.

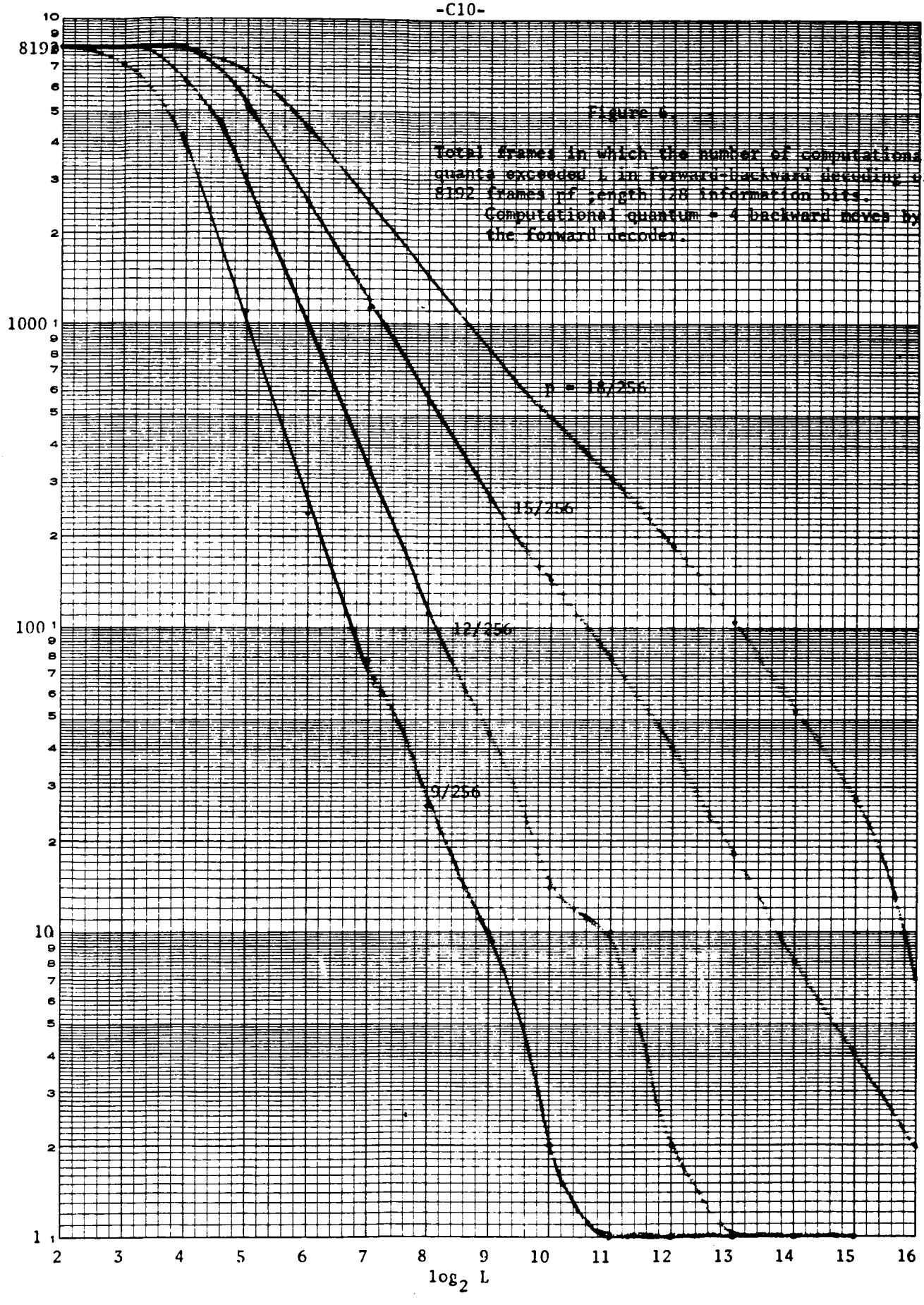


Figure 6.

Total frames in which the number of computational quanta exceeded L in forward-backward decoding of 8192 frames of length 128 information bits.
Computational quantum = 4 backward moves by the forward decoder.

Since the graphs are log-log, a Pareto distribution would appear as a straight line. Indeed, one can fit a straight line quite accurately to these curves in the upper ranges, and thereby obtain an estimate of the observed Pareto exponent. We believe that such estimates can be made to an accuracy of about ±.05. The observed exponents α_o appear in Table I. We also include for the forward scheme the theoretically predicted Pareto exponent α_t, and for the other two schemes, the values of $3/2\,\alpha_o$ and $\frac{3}{2}\alpha_t$, which are suggested by the analyses of Appendix B. Finally, we tabulate the frame overflow probability p_{of}, and the frame and bit error probabilities p_{ef} and p_{eb} in decoded frames.

TABLE I

Scheme	p	α_o	α_t	$3/2\alpha_o$	$3/2\alpha_t$	p_{of}	p_{ef}	p_{eb}
Forward	9/256	1.51	1.28	-	-	0	2.4(-4)	1.4(-5)
Forward	12/256	1.11	.95	-	-	0	8.6(-4)	6.7(-5)
Forward	15/256	.80	.65	-	-	6.1(-4)	5.9(-3)	5.4(-4)
Forward	18/256	.63	.49	-	-	3.2(-3)	1.6(-2)	1.7(-3)
Forward-Backward	9/256	1.85	-	2.26	1.92	0	2.4(-4)	1.4(-5)
Forward-Backward	12/256	1.51	-	1.66	1.42	0	7.3(-4)	4.7(-5)
Forward-Backward	15/256	1.07	-	1.20	.98	2.4(-4)	2.1(-3)	2.1(-4)
Forward-Backward	18/256	.79	-	.94	.74	8.6(-4)	1.0(-2)	9.9(-4)
Side-by-Side	9/256	2.17	-	2.26	1.92	0	0	-
Side-by-Side	12/256	1.66	-	1.66	1.42	0	2.4(-4)	-
Side-by-Side	15/256	1.21	-	1.20	.98	0	8.6(-4)	-
Side-by-Side	18/256	.88	-	.94	.74	4.9(-4)	3.8(-3)	-

The first thing to be noted about these results is the considerable disparity between the predicted Pareto exponent and that actually observed, the latter being considerably superior at each error probability. As this behavior is inconsistent with all the previous experiments of which we know, (except those on the Pioneer code; see the Interim Report [1966], pp. 41-42), it strongly suggests a defect in the simulation. We spent some effort attempting to isolate this defect, considering the following possibilities:

1. The frame length is only four times the constraint length and could therefore be too short. However, we ran a number of runs with a frame length four times greater (N=512) and got identical exponents.

2. Not all moves might be being counted. However, first of all, the program is set up so that all moves end with either a forward or backward transfer, involving one of two subroutines, so that each gets apparently identical treatment. Moreover, if a fixed percentage of computations were not counted, we would still observe the same exponent; it would be necessary that the percentage of computations counted decrease with the length of the run just so as to give a straight line on a log-log plot. This seems unlikely.

3. The random number generator could be misbehaving. However, in extensive test runs the first-order error probabilities were observed to be correct; correlations tending to cluster errors would hurt the decoder performance; so the only possibility is that some sort of anti-correlation was introduced which aided the decoder. This also seems unlikely.

Between the unlikelihood of our observed results and the unlikelihood of the possible explanations, we are unsure which to choose. Fortunatley, we can develop our principal conclusions without a choice. We must warn, however, that all our conclusions should be read in the light of the possibility of a basic fault in the simulation.*

Accepting the observed exponent as valid, we find that our hypothesis that the side-by-side scheme would have an exponent multiplier

*Note added in final typing. Further simulations show that the distribution assumes the theoretical exponent α_t for larger L and lower $Pr(C > L)$, and the exponent α_o observed in the lower ranges is due to counting computations per frame rather than per bit. Our results may therefore be believed.

of 3/2 is in complete agreement with the experimental results. At the extreme probabilities, observed performance is slightly below the predicted, though within the range of experimental error; possibly this is an indication of a real effect which sets in outside the neighborhood of $\alpha=1$. We consider these results a confirmation of the substantial validity of our horseback analysis methods.

The forward-backward simulations do not make quite as pretty a picture, but are not at all discouraging. The bulk, but not all, of the expected improvement in exponent was obtained at all probabilities but the lowest; the difference from the side-by-side values is large enough and consistent enough to be significant and not the result of experimental error. Secondly, the curves are quite close to ideally Pareto; there is only a hint in the lower regions of the two upper curves of the complicated double-knee behavior foreseen in Appendix B.

Let us consider possible explanations for these two discrepancies between prediction and observation. To explain the inferior exponent, the first possibility is that the backward code is considerably inferior to the forward. Figure 7 displays the distribution of computation measured for each; the backward code does have a larger coefficient than the forward, but only a marginally worse exponent (1.08 against 1.11), which would lead us to think that the same noise bursts were dominating the distribution, but that the backward code takes a longer search to surmount a burst than the forward; the exponent of the combined scheme should then still be $(3/2)\alpha$. A much more plausible explanation is that the pictures we drew of computational failure patterns were a little too simple; there is some spread in the lengths of the noise bursts which give a certain amount of computation, and some may exceed a constraint length; two slightly separated noise bursts are more than just twice as bad as one. Both these effects would break down the independence of noise bursts effects and tend to degrade the exponent, and this is probably what is happening. We shall return to this point below.

The simplest explanation for the failure to observe knees is that all our data fall in the regular part of the curve. To test this hypothesis we ran the forward-backward scheme at p=12/256 with the shortened codes of constraint length 24 and 15; we also ran forward

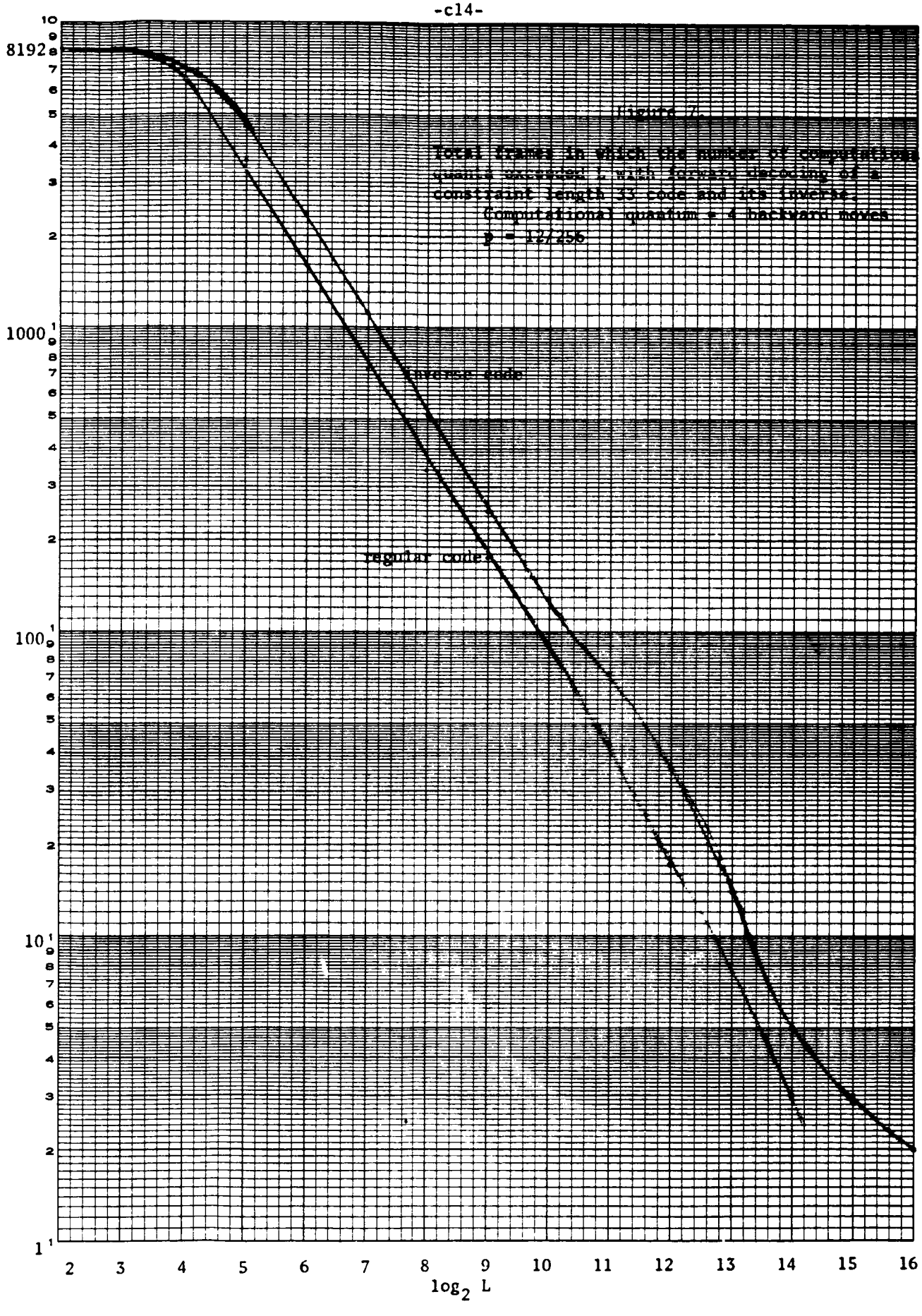


Figure 7

Total frames in which the number of computations exceeded L with forward decoding of a constraint length 33 code and its inverse.
Computational quantum = 4 backward moves
p = 12/256

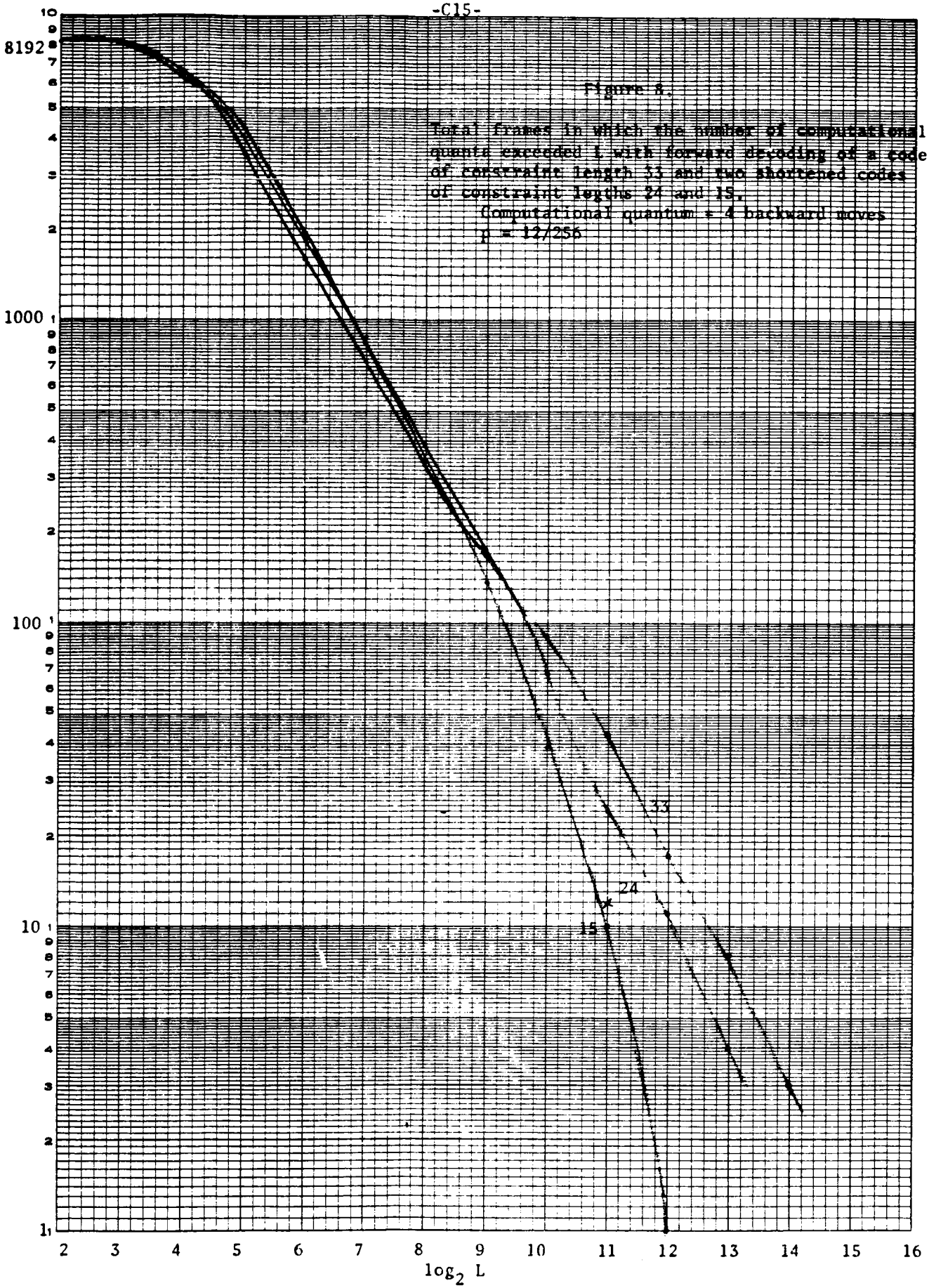


Figure 8.

Total frames in which the number of computational quants exceeded L with forward decoding of a code of constraint length 33 and two shortened codes of constraint lengths 24 and 15.

Computational quantum = 4 backward moves

p = 12/256

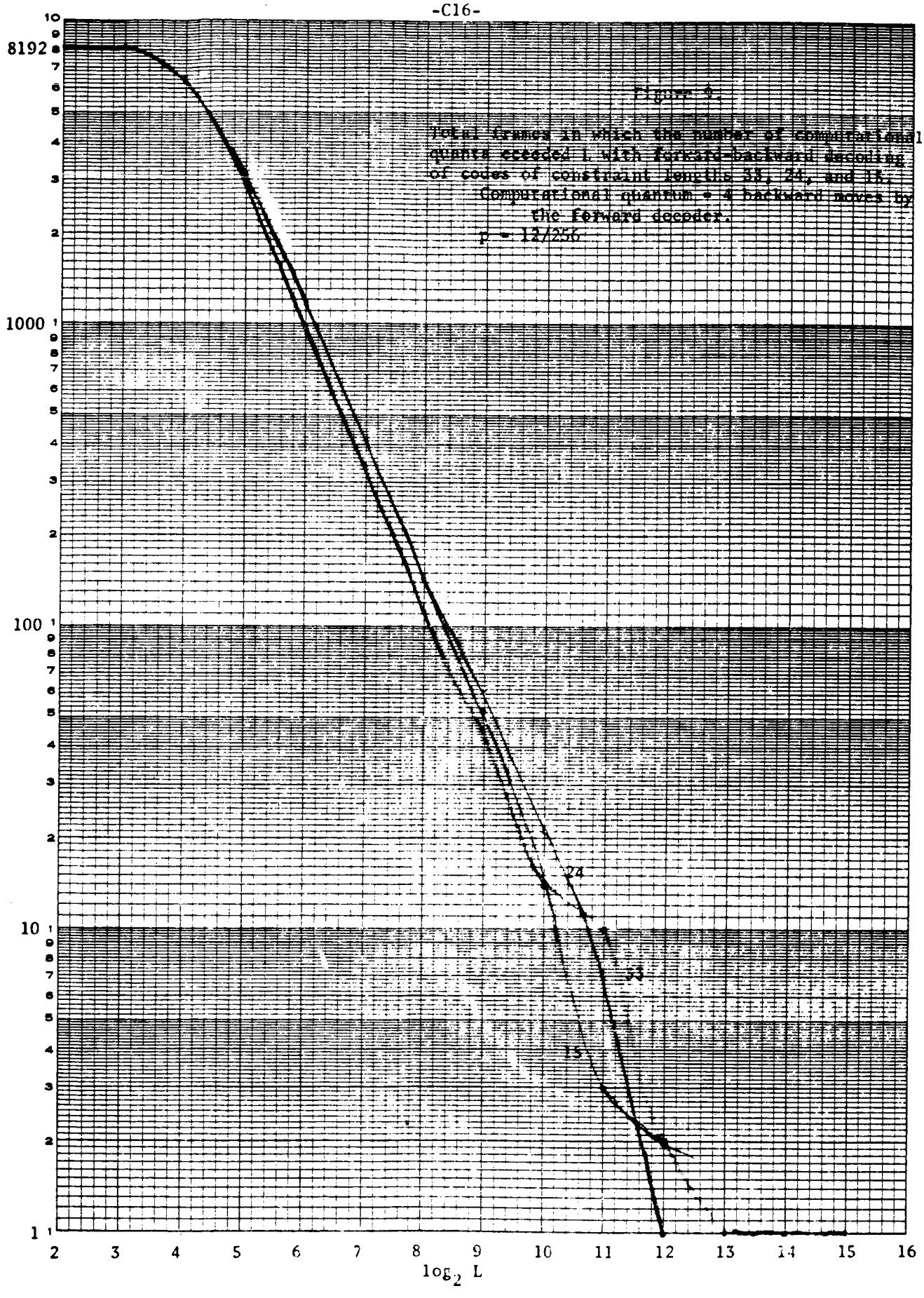


Figure 9.

Total frames in which the number of computational quanta exceeded L with forward-backward decoding of codes of constraint lengths 58, 24, and 15.
Computational quantum = 4 backward moves by the forward decoder.
$p = 12/256$

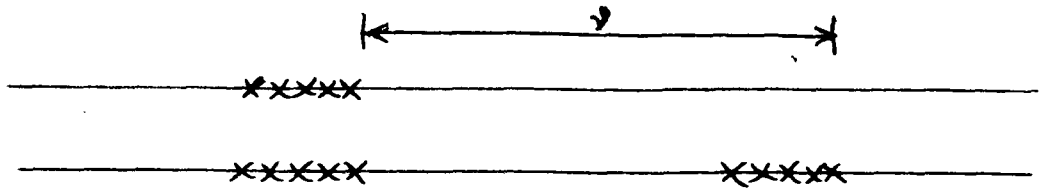
simulations as references. All other parameters were the same as in the previous simulations. The resulting distributions of computation are graphed in Figures 8-9; Table II records the associated probabilities or error. There were no overflows.

TABLE II

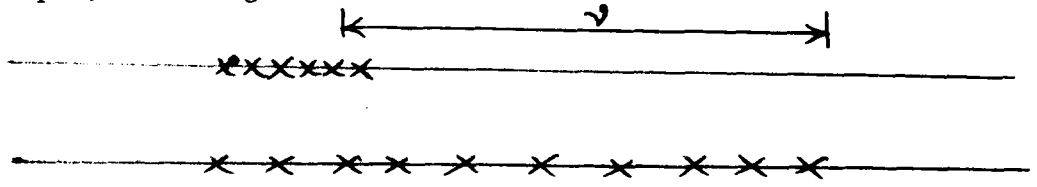
Scheme	Constraint Length	Pef	Peb
Forward	33	8.6(-4)	6.7(-5)
Forward	24	1.2(-2)	7.6(-4)
Forward	15	3.6(-2)	2.1(-3)
Forward-Backward	33	7.3(-4)	4.7(-5)
Forward-Backward	24	5.5(-3)	2.9(-4)
Forward-Backward	15	2.4(-2)	1.3(-3)

Inspection of Figure 8 shows that shortening the constraint length does not affect the ordinary distribution of computation, except that in the lower regions the beginnings of a knee are observed, due to many long searches being prematurely terminated by the occurrence of undetected errors. The backwards-forwards curves, however, are negligibly different from one another, except that the curve for the shortest constraint lengths begins to show a knee in the lower regions, which is explained by the previous sentence. No tendency for the curves to flatten out toward the original exponent α is observable above the knee; the observed curves are simpler than those predicted.

We conclude that our horseback analysis methods are neither far off nor totally satisfactory in this case. Most likely what is happening is that some noise burst not of the critical density, but peculiarly shaped for the maximum interference forward-backward decoding, is dominating the computational behavior, rather than the pattern



which we postulated as the dominant one. Such a pattern might be, for example, something like this:



In any case the existence of the former pattern establishes an upper bound of $3/2\ \alpha$, and this at least is consistent with the observed results.

Returning to Tables I and II, we can comment that the probability of undetected error seems to decrease only slightly in the concatenation schemes. This is consistent with the independent observation that the correlation between size of search and probability of error in the search is rather weak. It is true that the longer the search, the more probability of an undetected error, but there are many more short searches than long, and this tendency for error probability to decrease is mild for short searches. In Appendix A we hypothesized that the probability that a search of L computations would result in an undetected error, with a code constraint length of ν, would be approximately

$$1-(1-q^{-\nu})^{L} \simeq Lq^{-\nu},\quad L \le q^{\nu}; \tag{3}$$

this is the sort of weak dependence on L actually observed.

Let us proceed to consider the observed distributions more quantitatively. All the distributions are fitted very well by a two-segment curve of the form

$$Pr(C>L) = 1, \quad L \le L_o;$$
$$= (L/L_o)^{-\alpha}, \quad L \ge L_o, \tag{4}$$

where α is the exponent already determined. Thus, all that remains to specify the curve is the constant L_o which determines the coefficient and the break point. We see that, except at p=9/256, we have approximately $2^4 \le L_o \le 2^5$ for all curves. If we recall that the units of computation are 4 backward moves, then the distribution of backward moves C_b is

$$Pr(C_b>L) = Pr(4C>L) = 1, \quad L \le 4L_o;$$
$$= (L/4L_o)^{-\alpha}, \quad L \ge 4L_o; \tag{5}$$

the distribution of total moves, C_t, forward and backward, in a frame of length 128 is

$$Pr(C_t>L) = Pr(2C_b+128>L) = 1, \quad L \le 8L_o+128;$$
$$= \left(\frac{L-128}{8L_o}\right)^{-\alpha}, \quad L \ge 8L_o+128;$$
$$\simeq (L/8L_o)^{-\alpha}, \quad L >> 128\alpha. \tag{6}$$

Thus the coefficient is of the order of magnitude of one to two times the frame length, as has been consistently observed in sequential decoding simulations.

In the above, we have ignored the computations wasted on the undecoded pair in the side-by-side and backward-forward schemes, because if many information streams were encoded in parallel, this waste would become negligible. We have also, with less justification, ignored the overhead involved in time-sharing a sequential decoder between many parallel streams, simply because this depends so much on implementation.

With these exceptions, the coefficients for the concatenated and unconcatenated schemes are thus all of the order of a frame length, and to good quantitative accuracy, the distribution of computation per bit has a coefficient of nearly one:

$$P_r(C_{n,t} > L) \simeq L^{-M\alpha}, \quad L \geq 1, \tag{7}$$

where M is the exponent multiplier.

Comparison of Concatenated and Unconcatenated Schemes

In this concluding section we shall try to compare concatenated with unconcatenated schemes. We shall assume that per-bit computational distribution is given by (7) for both, that any multiplier M can be achieved with negligible rate loss by sufficient complication, that the constraint length is long enough to give a negligible probability of error, that the probability of overflow per bit p_o is given by

$$p_o = (\mu B)^{-M\alpha} \tag{8}$$

where μ is the decoder speed advantage and B is its buffer size, and finally that $M\alpha \leq 1$ so that the average computational load is dominated by searches just less than μB, and given by

$$p_o \mu B = (\mu B)^{-M\alpha + 1}. \tag{9}$$

(9) implies that

$$\mu \geq p_o \mu B; \tag{10}$$

taking the minimum value, we have the decoder parameters

$$B = p_o^{-1}$$
$$\mu = p_o^{\frac{M\alpha - 1}{M\alpha}} . \qquad (11)$$

Note that beyond the point where the buffer size is the inverse of the desired overflow probability, additional size buys nothing; for $\alpha < 1$ the decoder becomes average-speed-limited.

Assume that one wants to buy about one decibel over ordinary sequential decoding and thus wishes to use an α of about 1/2; suppose further that a bit overflow probability $p_o=10^{-5}$ is satisfactory in either case. (Actually, one would want a smaller probability for the concatenated over the unconcatenated scheme, since more bits would probably be lost per overflow with the former.) The alternatives are:

1. No concatenation; M=1, $B=10^5$, $\mu=10^5$;
2. Concatenation; M=2, $\mu B=10^5$.

At low bit rates, it is possible that the former is the easier solution. On the other hand, suppose one wants an additional decibel, so that α is about 1/4; then some alternatives are:

1. No concatenation; M=1, $B=10^5$, $\mu=10^{15}$
2. Some concatenation; M=2, $B=10^5$, $\mu=10^5$
3. Much concatenation; M=4, $\mu B=10^5$

In this case, some amount of concatenation is the only feasible approach.

We conclude that, although the brute force alternative of increasing the buffer size and decoder speed to their maxima will allow penetration considerable below R_{comp} at low bit rates, concatenation schemes will be necessary if extreme requirements are to be met.

Reference

Codex Corp., Interim Report on a Coding System Design for Advanced Solar Missions, Contract NAS2-3637, Watertown, Mass., October 20, 1966.